

Building a Real Time Bus Tracking Data Display System

Jason C. Dudley

A Capstone Project Submitted to the
University of North Carolina Wilmington
in Partial Fulfillment of the Requirements for the Degree of
Master of Science

Department of Computer Science
Department of Information Systems and Operations Management
University of North Carolina Wilmington

2011

Approved By Advisory Committee

Dr. Ron Vetter, Chair

Dr. Jeff Brown

Dr. Tom Janicki

Table of Contents

1. Introduction.....	3
2. Background and Related Work.....	4
3. Design Methodology.....	6
4. Project Risks and Related Constraints.....	9
5. Implementation	10
5.1. Application	10
5.2. Design Issues.....	12
5.3. Software and System Quality Metrics.....	13
6. Results.....	14
6.1. Design Methodology.....	14
6.2. Website.....	15
6.3. LED Program.....	17
6.4. ETA Calculation.....	18
7. Summary.....	23
8. Bibliography.....	25
9. Appendices.....	26

1. Introduction

Real time tracking is becoming more and more popular as devices utilizing the Global Positioning System (GPS) become more readily available. There are a plethora of different uses for this technology from keeping track of loved ones to guiding cruise missiles to their target. Wave Transit, the local bus service provider for Wilmington North Carolina, has a system in which the buses send their coordinates (latitude and longitude, among other data) to a central database every thirty seconds using AT&T's cellular data service. This data allows the dispatcher to know where all the buses are at any given time. The data has previously been used on the Wave Transit website to show the current location of specific buses. Because Wave Transit opened a new central bus station they wanted to make this data available to the passengers who come to the station in a way that benefits them.

The goal of this project is to enable passengers to easily access data related to the bus that they are interested in. This goal was accomplished by creating three display monitors at the new Wave Transit station. Two of the displays show the estimated arrival times of the six buses that come to the station. The estimated arrival times are being calculated by a program developed by Dr. Jeff Brown in the Mathematics & Statistics Department at the University of North Carolina Wilmington (UNCW). These displays also serve as a way to provide current news and information about Wave Transit. For instance a message can be displayed to broadcast bus downtime and any emergencies that may arise quickly and easily. The news and information is updated using a content management system by Wave Transit employees. The third display shows a real time map marking each of the buses with a label that identifies which bus it is.

Another goal of this project is to make all of this real time data available to the public on the Wave Transit website. The website displays a list of all available buses and users can click on and view a real time map for each bus. An information window on the map shows users the route, next stop, estimated time of arrival (ETA), and the buses' direction. These pages had to be made in a way that is mobile friendly so passengers can pull up maps on their mobile device while they are at a bus stop and watch the bus make

its way to the stop. Since many users have smart phones, the website detects which mobile browser is requesting the information and formats the information for the smaller screen size. In addition, the website centers the map on the bus's location and follows it as it moves through Wilmington.

The data presented on the displays inside Forden station is invaluable to the passengers that it serves. They are able to quickly and easily realize the amount of time until their bus will be at that particular stop. Having this information readily available should help to increase ridership within the Wave Transit service area. In addition, the perception of Wave Transit's bus service has improved due to the fact they have this technology available. It seems that many major cities have this technology and some have just been released as recently as 2010 [6]. Because approximately 25% of Wave Transit's ridership is directly due to the university [3], as the technology becomes more widespread UNCW ridership should also increase. The reason for this is a student can quickly see how soon a bus will be at a particular stop and he/she will be able to plan accordingly. The public benefits from this project because the project is showcased at the bus station and the Internet for all the public to see.

2. Background and Related Work

Currently several large cities have similar bus tracking applications in place. The New York Metropolitan Transportation Authority (MTA) recently released a pilot project on a single route in Brooklyn called MTA BusTime. They have created a web page which displays this route with information on each stop showing how far each bus is from that stop, a bus icon is also displayed to show where each bus is on the map. This web page has been developed in a way that it is mobile friendly as well, allowing users to access it at their convenience. MTA updates the map with new data every 30 seconds, much like Wave Transit's system does. This pilot has not been live long enough for the MTA to determine if it is increasing the amount of ridership on that route [1].

Regionally, both North Carolina State University (NCSU) and the University of North Carolina in Chapel Hill (UNC) have bus tracking applications. The NCSU

application is provided by a company called TransLoc®. TransLoc® offers a real time map, at the bus stop, on the web, and anywhere via mobile application. The map shows the relevant bus routes, stops, active route color key, and real time bus location. Each route is color coded and has a marker at every stop that is clickable. The information window that displays after clicking a marker shows what buses stop at that location and the estimated arrival time of that bus. The map updates every second so you can see the buses' actual location in smaller increments. This is a unique feature that this particular application has that is of great benefit to the user because the bus location is more dynamic. On the mobile website the available routes are shown first, once a route is selected the map is shown with only that route overlaid on the map. This application focuses solely on the map to provide information about the bus schedule [9].

UNC uses an application provided by NextBus® Incorporated [8]. The application is not centered on the map, instead it uses a form with dropdowns that enables the user to choose a route, direction and stop. After entering this information the ETA is provided for the next bus in that route to reach that stop. On the final page an option for viewing a Google map is given. The Google map draws the route with all the stop locations, each stop location is clickable providing an information window on the route, direction, and next departure time. This application offers the ability to hide or show the vehicles and stops on the map using a checkbox. The other unique feature is the ability to select more routes to be shown on the map. There is not as much focus on mobile users in this application as it just offers an image of the map with the bus location noted with two blue lines [7]. The user interface is lacking as it is tedious to select the correct route, and the stop and destinations, especially if you are not familiar with the area.

Wave Transit quickly saw the value in texting and has already implemented short message service (SMS) capability to receive real time bus location information. SMS is available on over 98% of all cell phones which makes this service available to almost any passenger with a cell phone [11]. Passengers can simply text "Bus 101" to 90947 to receive that buses next stop. UNCW routes are a little different where passengers can text "Bus R" and get the next stop of the bus on the UNCW Red Route. When a passenger sends a text message to 90947 it is sent from their service provider to an SMS aggregator

who then passes it on to a content provider (in this case Mobile Education LLC is the content provider for the short code 90947). Mobile Education has server software that allows it to make a database query against the Automatic Vehicle Location (AVL) database which holds the buses next stop. The content server then replies with the information from the AVL database to the SMS aggregator who sends it back to the passenger's phone using their service provider. This system allows a passenger to quickly and easily find out where a bus is in any given route, but with low accuracy because it only shows the next stop. The new application, developed as part of this project, improves on that by showing a map of the bus with ETA to the next stop, however, it will only be available to smart phones.

Another service that Wave Transit offers to its passengers is Google Transit. Google Transit is a public transportation planning tool that combines public transit data such as stop, route, schedule, and fare information with the power of Google maps [12]. A passenger interacts with this service through the normal Google maps interface which is already available in 12 different languages and is mobile friendly. After going to Google maps passengers simply get directions from one place to another in Wilmington. Google offers a public transportation option as a mode of travel along with each step the passenger would need to take to get to their destination. Fellow graduate student, Mr. Andy Hermann is currently contracting with Wave Transit and has recently finalized the route data to Google Transit. With this data Google can perform the necessary calculations to offer it as a transportation option in Wilmington. There are several files that need to be submitted to Google to make this work, but they are all publicly available at this URL http://www.wavetransit.com/gtransit/google_transit.zip.

3. Design Methodology

The project had a time constraint in which basic functionality for the displays had to be ready by the grand opening of the bus station in April 2011. Due to this time constraint the extreme programming paradigm (XP) was chosen initially. The XP model would allow the development to begin immediately to get a prototype to the client as soon as possible. The XP model advocates frequent "releases" in short development

cycles [2]. This would require tight communication with other members of the team as well as the customer. The coding and functionality would ultimately drive the advancement of the project as the customer doesn't know exactly what to ask for. This is common in cutting edge applications where only the prototype can help the customer understand what is possible and ultimately spark functionality requests. The scope of the project would obviously be at risk but, the deliverables should define a clear goal and anything beyond that would be a bonus to the customer. This seemed like a good fit for the project but due to the risks it was not chosen.

The Spiral model (Figure 1) seemed to be most useful for this project. Although it is said that the spiral model is used for large, expensive, and complicated projects it can be of use for smaller projects as well [4]. The system requirements were very clear in this project and were well defined in the very first step. The preliminary design was implemented for the opening of the station. This strayed from the definition of a Spiral model only slightly in that the first prototype was operational. From the preliminary design useful feedback was gathered from the customer and passengers to evolve the project into its second prototype. At that point the next iteration began by evaluating the first prototype, defining new requirements, planning, designing, and finally building the second prototype. This was an iterative cycle until the conclusion of the project. This allowed enough time to complete three full iterations.

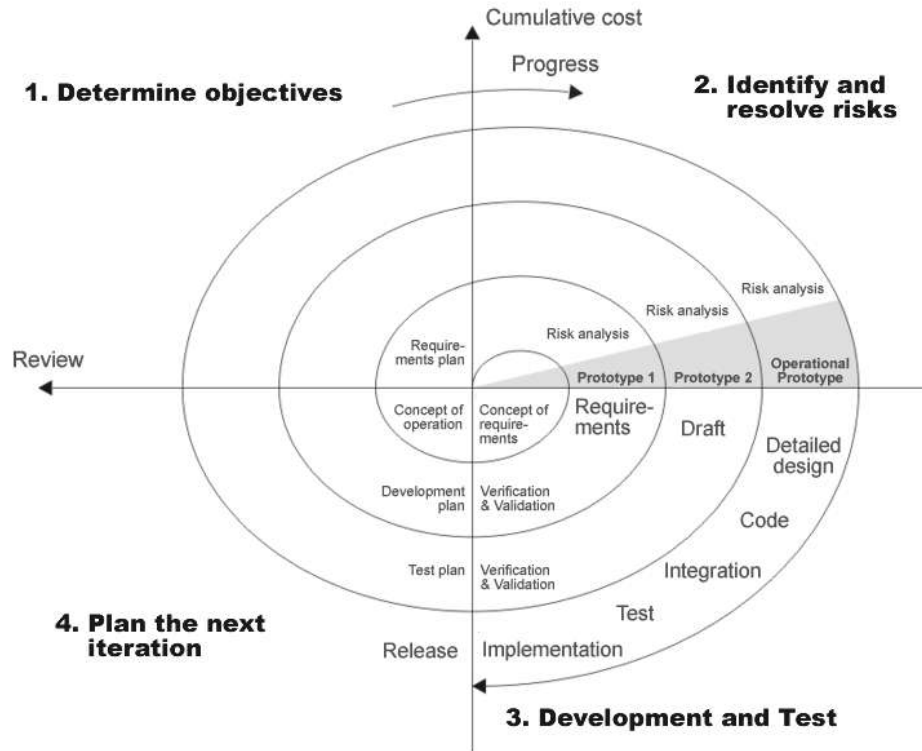


Figure 1: Spiral Model (Created by Conrad Nutschan)

Several use cases were developed to help define what features the system should have. The use cases also help to identify potential problem areas and how best to fix those problems. Use case diagrams make it easy to see how a user would interact with the system and the options they have depending on the method in which they choose to use the system. Appendix A has five use cases that have been developed as part of the requirements planning. The use cases are named after the method that the passenger interacts with the system, passenger via computer, passenger via Mobile, passenger via station, passenger via SMS and there is one use case for a Wave Transit employee. The user is the primary actor on the system where the goal is for the system to provide the user some sort of information about the Wave Transit bus system. The user is broken in to four different actors to show the different methods of accessing the system and resulting features they will be able to utilize. There is one secondary actor that is an employee of Wave Transit that will be responsible for entering data into the system. The employee will login to the system using a username and password and then be presented

with the news page that they have permission to update. The initial use cases were brief and to the point, keeping them simple made it easier for the customer to understand but still clearly defined the functionality.



 Route	 ETA
101 BROOKLYN	21 mins
102 UNIVERSITY	21 mins
103 CENTRAL	21 mins
104 EAST	21 mins
105 MEDICAL CENTER	21 mins
106 WEST	21 mins

Figure 2: Initial ETA Display

4. Project Risks and Related Constraints

In every project it is important to identify the risks and constraints before agreeing to a contract. The first step was to identify the stakeholders, many people and organizations were affected by the system. As the project was initiated the resulting stakeholders were identified as: Wave Transit, Hook Systems, Dr. Ron Vetter of the Computer Science Department, Dr. Jeff Brown of the Mathematics Department and the passengers of Wave Transit. Wave Transit is the most important stakeholder they provided the funding for the project and could have ended it at any time. Hook Systems was responsible for the internet connectivity, server setup, displays and the local area network to run the light-emitting diode (LED) displays. The work they did at the station was integral to getting the displays working when the station opened. Dr. Ron Vetter was responsible for a Java component that communicates with the LED displays to show the

ETA for each of the buses. Dr. Jeff Brown was responsible for a Java component that will estimate the ETA of the buses and make that available for other components to use. Finally, the passengers make use of the system making it easier for them to schedule their activities around the buses real time schedule. Communication between stakeholders was important to reduce the amount of risk in the components and infrastructure being setup in a way that everything works as one unit.

Several constraints were present in this project as well. The first, most obvious constraint was that the displays had to be ready before the new station opened to the public. That only allowed a few weeks for the development, testing and deployment. The second constraint was that the components were all developed in Java. This is the current system's primary development language and it had to be used for maintainability. In the future this project may be enhanced to provide even more functionality. The final constraint was that the system is available 24/7 to the public. This means that all components handle any errors that may occur and recover from them autonomously. The constraints were an important part of this project and by meeting them the project exceeded Wave Transit's expectations.

5. Implementation

5.1 Application

The system involves many different parts that worked together to accomplish the requirements set forth by Wave Transit. The source of all the data that is presented originates in the AVL Database. This database is where the buses send data every 30 seconds. The buses send information such as longitude, latitude, heading, route, and speed. From there the data is accessed on two different web servers. The estimated arrival time server grabs the data and does calculations on each bus and outputs a comma separated string to a specific URL (<http://mymobed.com/map/all.jsp>). The second server uses the AVL data to map the buses on a real time Google map. This map is accessible via the following URL (<http://www.wavetransit.com/RealTimeBusTracking.aspx>).

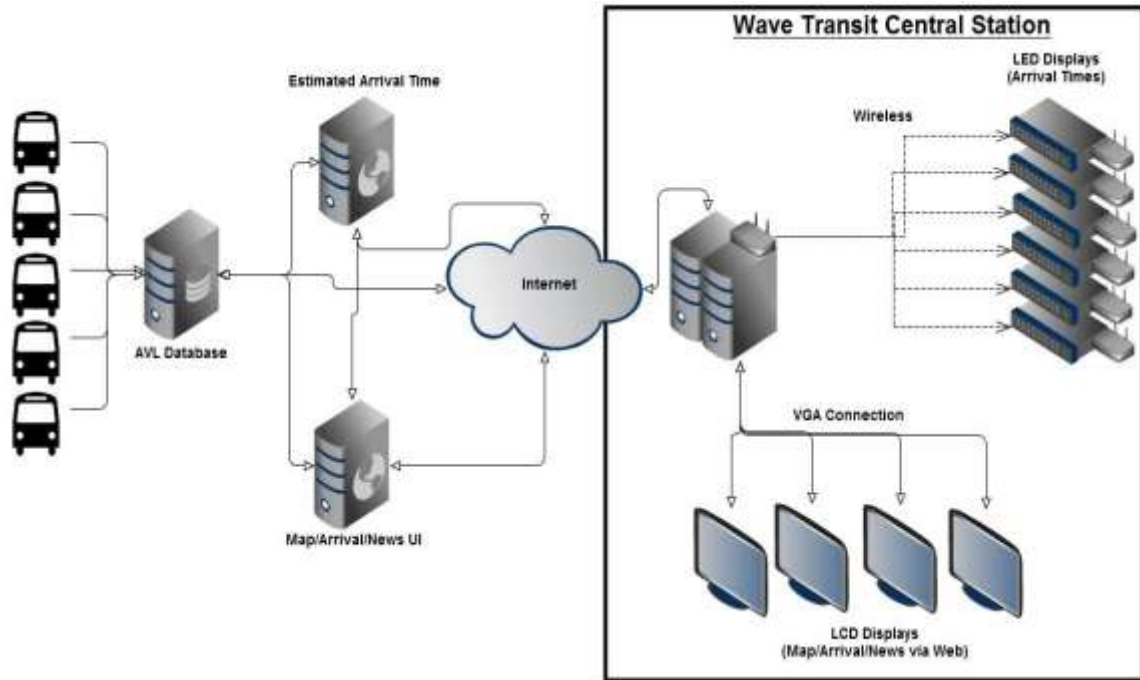


Figure 3: Implementation Diagram

The second server also communicates with the estimated arrival time server to request the ETA string and parse it to display in a table that is presented to the user via another URL. The server inside the Wave Transit station can access all the content from both of the servers via the Internet. There is a Java application running on the inside server that makes a request for the ETA string, parse it, and sends the times to each specific LED display, mounted above each buses' stop. The Java application opens sockets to each LED display individually and sends the appropriate arrival time. The communication uses the wireless local area network at the station. Two PCs control two of the liquid crystal displays (LCD) via dual video graphics array (VGA) ports. The displays use a "spanning" configuration in Windows to show two different web pages displaying either ETA or real time map in full screen mode. The other PC controls the other display in the same fashion.

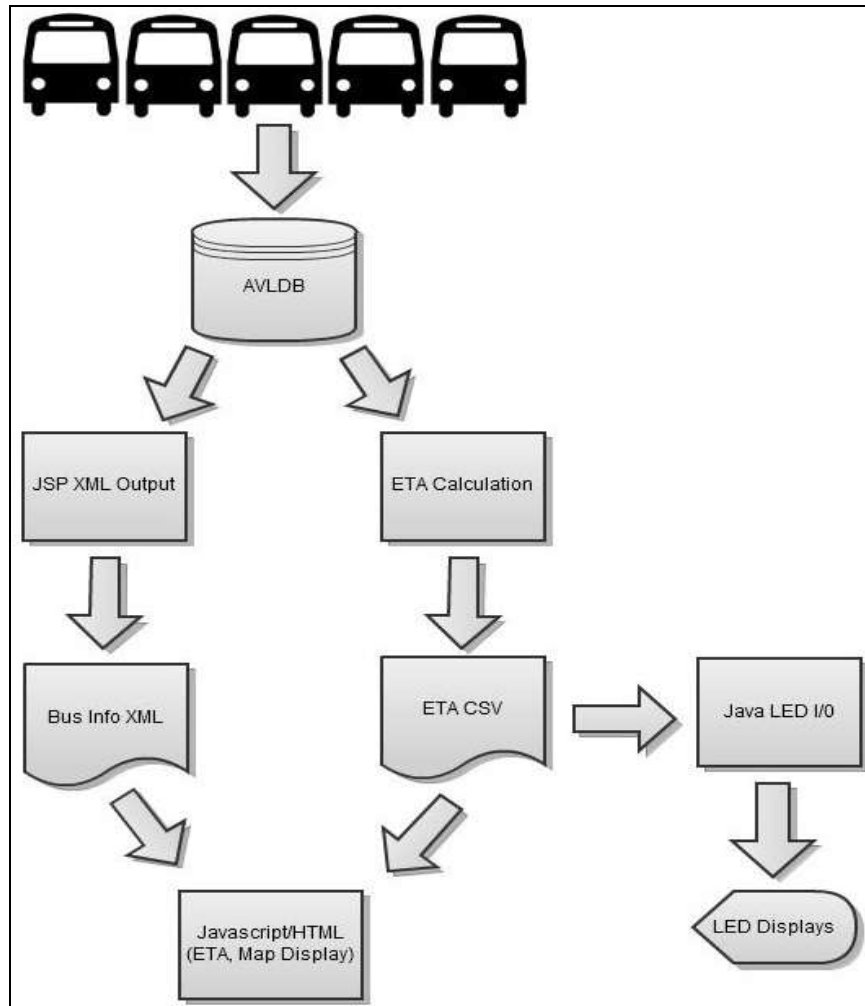


Figure 4: Data Flow Diagram

5.2 Design Issues

There were a few issues that arose when going over the project requirements. The first issue to overcome was the programming language; lack of experience in Java Server Pages (JSP) presented issues that slowed down the projected implementation of this project. This was a requirement so that the system can be maintained in the future by Dr. Vetter. JSP was released in 1999 by Sun as a direct competitor to ASP and PHP. JSP helps to bridge the gap between Java and the web. Throughout the project the syntax and integration with Java presented a steep learning curve. I was able to become proficient in JSP by learning the best practices, so now it can be used in future projects.

Another issue was the presentation of the data to the user. At the bus station the data on the page changes without refreshing the entire page. Previously a meta refresh was used to update the information. That wasn't acceptable because it would cause a noticeable blink of the user interface (UI) every 30 seconds. Since JSP is a server side language the only way around this was to change the data using a client side language like JavaScript. Also, the displays are all 1080p so all of the interfaces needed to be optimized for that resolution. This would have represented a challenge for the news page that was part of Wave Transits public site. The public site should be accessible to all resolutions which means it cannot be optimized for one of the highest resolutions (1080p), instead it is optimized for the more common resolution of 720p. The news page was eventually merged with the arrivals display in prototype 3 so, this concern was eliminated.

5.3 Software and System Quality Metrics


The International Organization for Standardization (ISO) defines quality as "the totality of characteristics of an entity to bear on its ability to satisfy stated or implied needs". [10] This definition itself is very vague, but in essence the quality will be measured by how well it conforms to the requirements and that it can be used as it was intended. In other words this project would be considered successful if it meets the scope in the allotted time, satisfies the customer (Wave Transit), and reaches the ultimate goal of providing a benefit to the passengers of Wave Transit. Meeting these broad goals ensured that the system met the requirements set forth by Wave Transit.


The software was developed based on best practices expressed for source code quality including readability, maintainability, low complexity and robust error handling. Because readability and maintainability go hand in hand, the software was written in a way that another individual can easily understand and update it if necessary. The software was broken down into individual components, each focused on a particular and singular task so as to have the least complexity possible. Finally, robust error handling was enforced to have a system that was capable of running 24/7 and provide the user with a system that provided them with the tools they needed to operate it successfully.

6. Results

6.1 Design Methodology

The spiral model proved to be an ideal way to implement the Wave Transit real time bus tracking system. Each prototype allowed for enhancement visually and functionally. As soon as prototype one was released for the public several suggestions came up for improvement in prototype two. For example, on the arrivals display inside the bus station (www.Coupons2YourPhone.com/bustracking/wavetransit/arrivals.html) a clock, date, and message of the day were all added in the following prototypes to increase usability for Wave Transit passengers. In addition, the map display (www.Coupons2YourPhone.com/bustracking/wavetransit/stationmap.html) was improved with a better orientation on the screen to show more of Wilmington and a "you are here" marker was added to show passengers where they were located while looking at the map. Along with these improvements we were also able to spot a few bugs in prototype one that was remedied in prototype two. One issue was a caching issue specific to Internet Explorer where it would cache the location of the buses which caused the map display to show the buses in the same location on each 30 second update.

The image shows a digital display for bus arrivals. At the top, the word "ARRIVALS" is in large white letters on a blue background. To the right is the Wave Transit logo and a digital clock showing "8:27 PM". Below this is a table with three columns: "Route" (with a bus icon), "6/9/2011", and "ETA" (with a clock icon). The table lists six routes: 101 BROOKLYN (2 mins), 102 UNIVERSITY (60 mins), 103 CENTRAL* (3 mins), 104 EAST (2 secs), 105 MEDICAL CENTER (4 mins), and 106 WEST* (3 mins). At the bottom, a small alert message states: "Alert: 101 BROOKLYN is currently undergoing maintenance. It will be up soon. Check back for more updates."

 Route	6/9/2011	 ETA
101 BROOKLYN		2 mins
102 UNIVERSITY		60 mins
103 CENTRAL*		3 mins
104 EAST		2 secs
105 MEDICAL CENTER		4 mins
106 WEST*		3 mins

Alert: 101 BROOKLYN is currently undergoing maintenance. It will be up soon. Check back for more updates.

Figure 5: Final Arrivals Display



Figure 6: Final Maps Display

6.2 Website

The website implementation started in prototype two as a simple upgrade to the existing system that used Yahoo maps. The only option was to choose a single route and have those buses show up on a Google map. Google maps allowed for the passenger to interact by clicking the bus marker and seeing information about that bus. Previously this information was shown above the Yahoo map image. Another improvement was to show all buses in the same route. The old map would only show a single bus on each route and the user had no control over which bus was shown. In prototype two, the user could see all the buses on any given route as well as the information specific to each bus. The text on the real time tracking page before was quite lengthy and may have distracted users from the pages main purpose. In prototype two, the page was made much shorter by moving the SMS texting instructions to its own page and providing a link for users to navigate to the page if they were looking for that information. The intent of the real time tracking page is much clearer now as the route selection interface is very prominent on the page and the use of bright hover states draws the user's attention.

Prototype 3 was a huge improvement over prototype two with the implementation of a mobile friendly tracking page as well as the implementation of the other use cases. Now a user can select from 3 options on the website, a table display of a single route, a map display of a single route, or all of the routes on a single map. The mobile friendly

version has a very simple user interface to cut down on load times. It has the option to view a table which is a text based display of a specific route or view a map of a specific route. Due to the large number of buses to display, a map with all buses was left out of the mobile version because it was unusable on a small screen. Along with all these improvements a simple implementation of Google's direction service was used to give the user an ETA for the bus to arrive at its next stop. This information is available along with route name, heading, speed, and next stop on all versions of the map and table display.

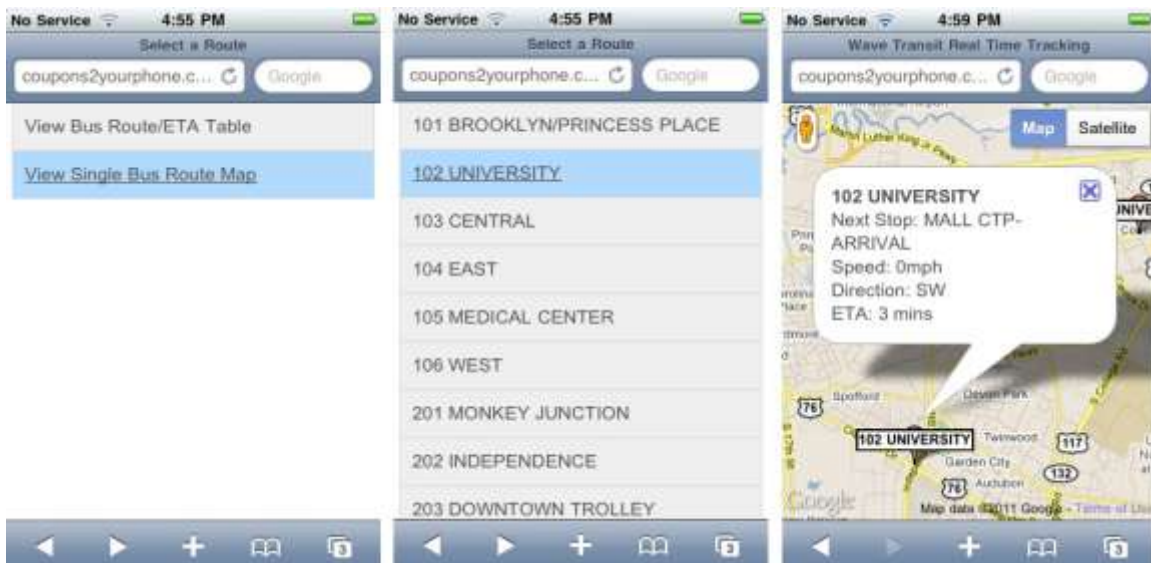


Figure 7: Final Mobile Implementation

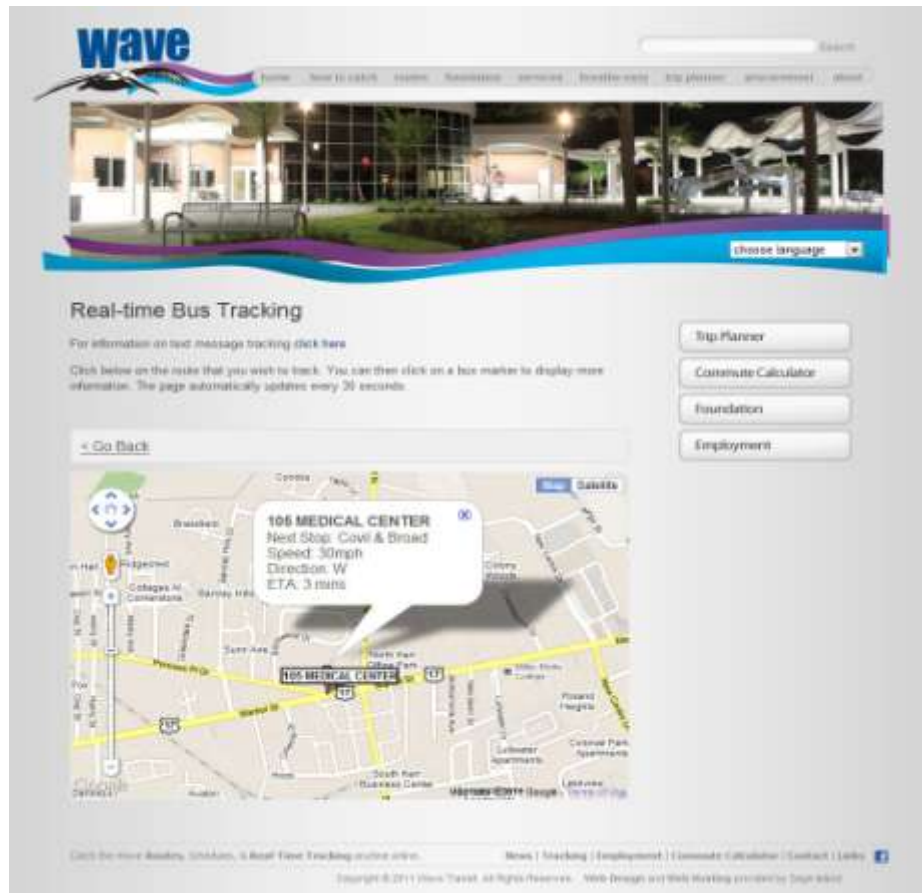


Figure 8: Final Website Implementation

6.3 LED Program

The final implementation of the LED Java code resulted in a shell program that calls a Java program twice every minute. Since CRON has a minimum scheduling resolution of 1 minute, it must call the Java program, sleep for 30 seconds, and then call the same program again. The Java program requests the ETA values from Mobile Education's server for each of the six routes then sends a command to the appropriate display to show that routes name and ETA. Each bus has a specific LED sign that it stops under which has a static IP address. The Java program the IP addresses are hard coded to display appropriate information on the correct LED sign. When the Java program is executed outside of normal business hours it simply sends a command to clear the display so nothing is shown (see Appendix C).

In addition to the Java that was written to display information on the LED displays, a UNIX shell program was written for a Wave Transit employee to execute a variety of commands on a single LED display through a command line interface using SSH. SSH is a network protocol that allows data to be exchanged using a secure channel between two networked devices [13]. The shell program has a number based menu that is very easy to use and explains each option that is available. When an employee logs in using SSH they are presented with a menu where they can stop or start the automated LED Display program as well as choose from more advanced options. Some of the advanced options include: set RTC time, set RTC date, set brightness, get display configuration, get error status, and execute an LED test. A user guide was provided to Wave Transit to show them how to access the program if they ever have a problem.

6.4 ETA calculation

Dr. Brown's ETA estimate algorithm calculates the ETA for the six bus routes that stop at Forden Station. This estimate is shown in real time on the LED and LCD displays at Forden Station to give passengers an idea of when their bus will arrive. Additional data from the AVL database was needed to calculate the ETA. Not only does the AVL database store information about the buses current statistics, it also has tables that contain information about each stop on a route as well as coordinates along a route called breadcrumbs. They contain the latitude, longitude, heading and timestamp information at different points along a given route. Each route has over 1000 breadcrumbs collected by driving the route and gathering data, but not stopping to pick up passengers. So, the total time to breadcrumb a route could be half the time it takes the bus to complete the route. To account for this time difference an error is added to the calculation after all the remaining breadcrumbs are added up. This error factor is only added if the estimate is more than five minutes. If the estimate is less than five minutes then there are not many, if any, stops left so the breadcrumb times should be accurate. The formula for the error calculation can be seen in Appendix D.

Data collection was performed on the six routes to determine how close to the actual time the estimates were. After collecting data for a full Saturday which included

about 14 cycles for each route, the average of actual ETA minus estimated ETA was 465 seconds with a median of 188 seconds over all routes at any given time. I expected the average to be quite a bit higher than the median because there are instances where the bus is at the station and the actual is at zero while the estimate has begun calculating the next cycle. This is how Dr. Brown coded the algorithm, if the bus is at the station it should start estimating when it will be back again. A median of 188 seconds is a closer estimation of how accurate the estimate is because the outliers are not factored in. I think this is an acceptable range for the buses ETA. From the data, we see that the estimates are more accurate as the bus gets closer to the station. So, when the bus is an hour away it would be acceptable to be off by a few minutes. The ETA serves its purpose and I would consider it close enough to the actual for a passenger to plan accordingly. As figures 9 to 14 show, some routes are more accurate than others. They ranged from a median of 285 seconds on route 102 to a median of 120 seconds on route 101. With this in mind it may be possible to adjust the error in Dr. Brown's algorithm for the routes with a higher median to make the ETA more accurate overall.

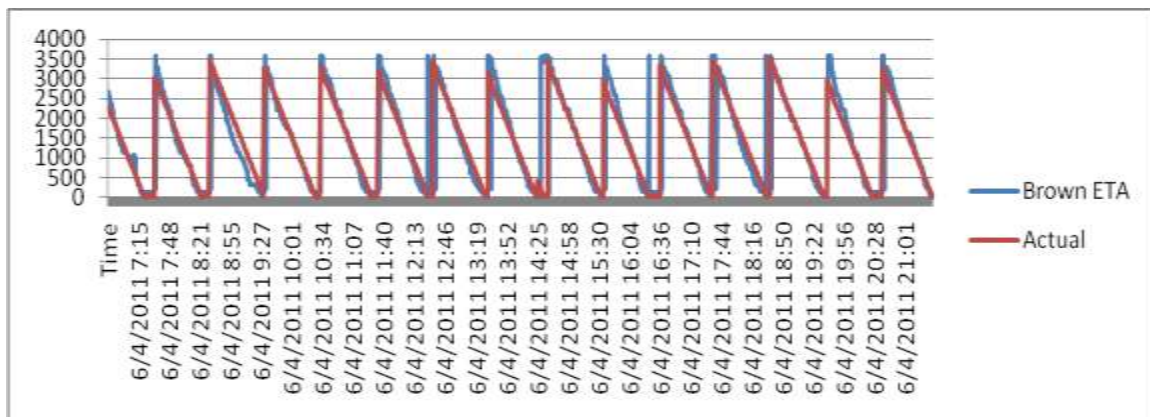


Figure 9: 101 Brooklyn (Average 261 seconds, Median 120 seconds)

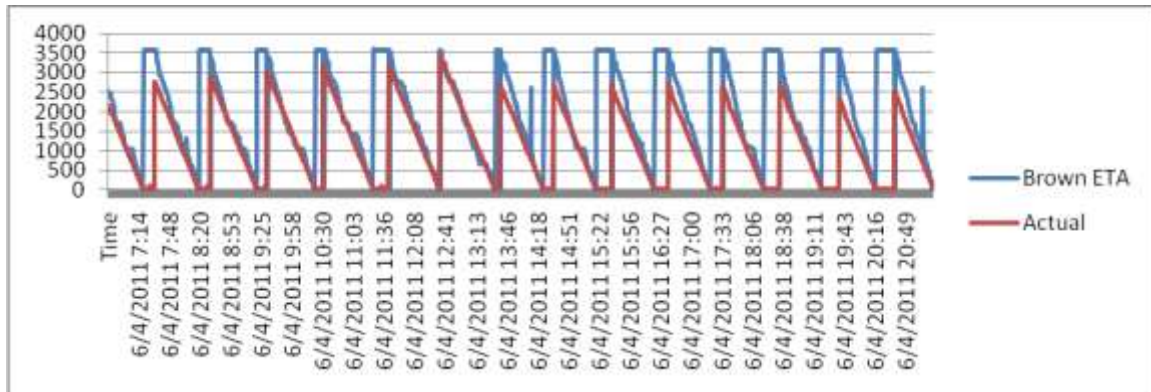


Figure 10: 102 University (Average 969 seconds, Median 285 seconds)

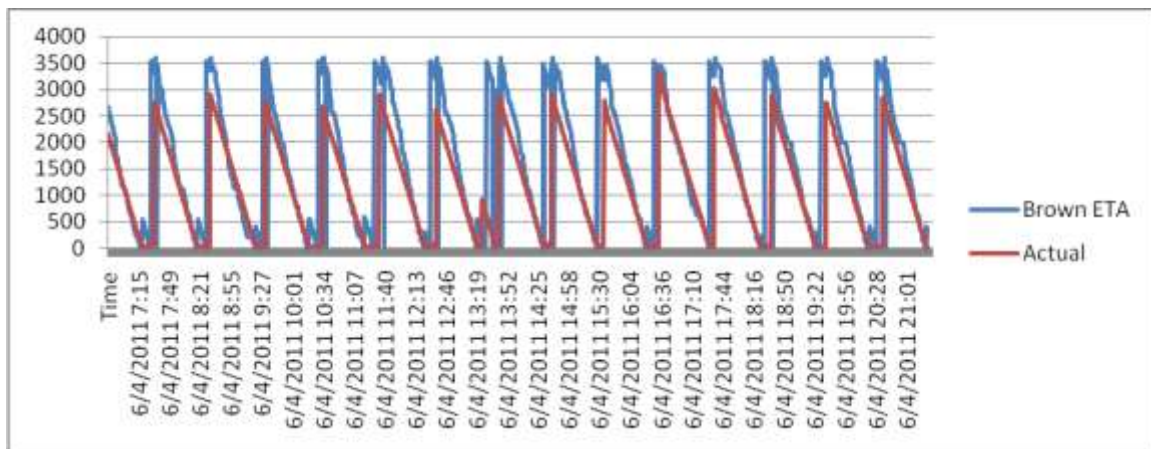


Figure 11: 103 Central (Average 576 seconds, Median 271 seconds)

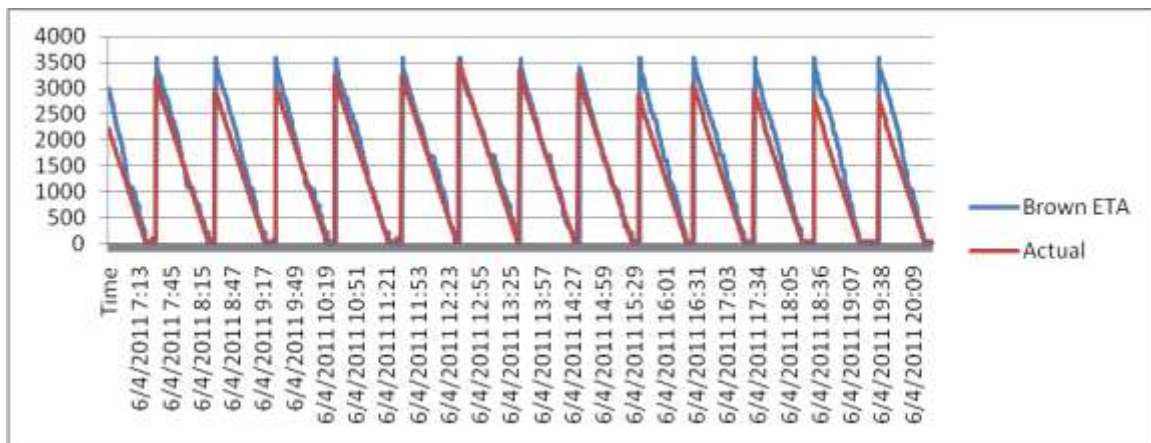


Figure 12: 104 East (Average 258 seconds, Median 162 seconds)

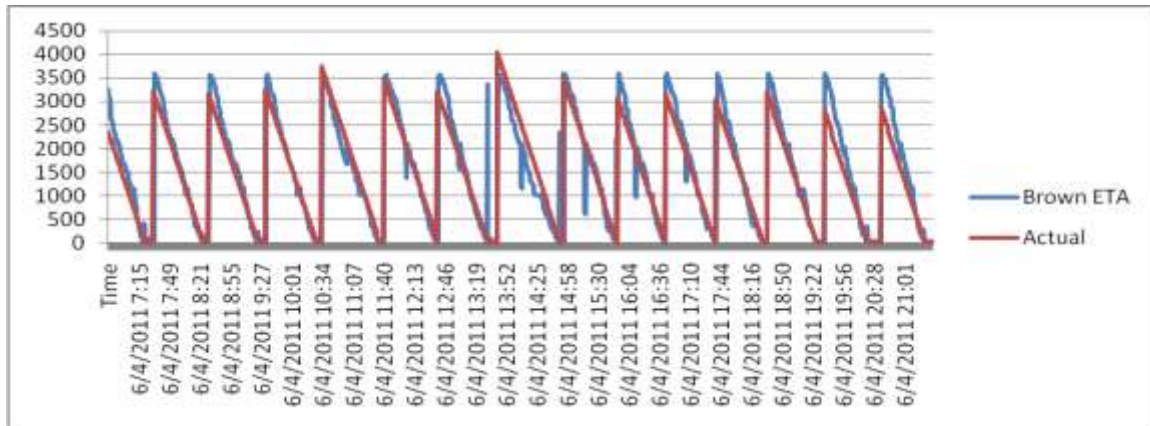


Figure 13: 105 Medical Center (Average 278 seconds, Median 158 seconds)

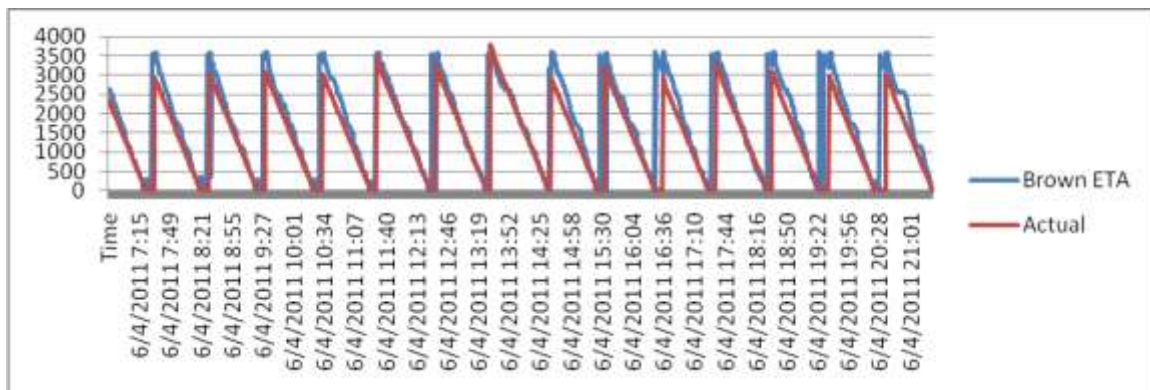


Figure 14: 106 West (Average 436 seconds, Median 202 seconds)

One interesting question which arose was, “How do these results compare with what Google’s direction service?” Starting with a simple implementation of Google’s direction service which determines the ETA to the next stop on Wave Transit’s map interface, a more complex program was written to determine the ETA to Forden Station. Google’s direction service allows for waypoints to be passed along with a directions request to keep the directions along a specified route. As the bus moves along its route waypoints are removed after the bus has passed them so they no longer affect the ETA. This would make sure that the ETA that Google returned would at the very least account for each stop before it would arrive at Forden Station.

After implementing the Google ETA for Forden station, data was collected to figure out how precise the ETAs values were. A simple program was written using JSP,

JavaScript and HTML. Every 30 seconds the JavaScript would get Dr. Brown's ETA, latitude, longitude and make a call to Google directions for an estimated ETA. Once the data is collected it sends it to a JSP file which executes an insert into the ETACompare table in the database. The data was logged from 8am to 5pm so there would be several iterations of the bus leaving and arriving at the station. After the data was collected, it was updated to record when the bus had arrived at the station. The bus arrival at the station was determined by drawing a bounding box around the station using the maximum and minimum latitude and longitude the bus could be in and still be at the station. From there it was an easy update statement to set an "arrived" bit field in the table to one when the bus met the specifications. After the update statement had run, the actual time of arrival could be determined by subtracting two dates for each record. The date would have to be analyzed in subsets so I would only be dealing with one circuit at a time. The first date comes from when the bus arrived during the current circuit and the second date was the current record that is being analyzed within that subset. After the actual time was computed for each record I could then graph the results to see how close the Google ETA was to Dr. Brown's ETA.

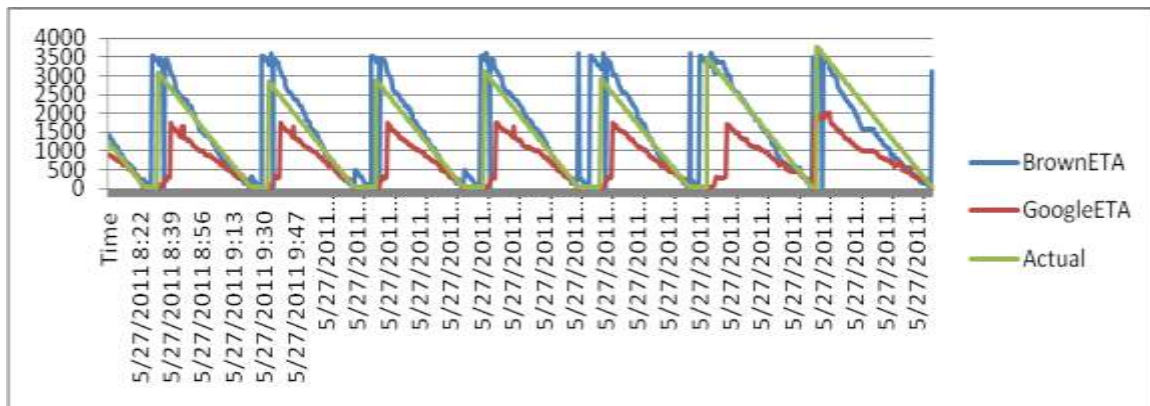


Figure 15: 103 Central (Google/Brown)

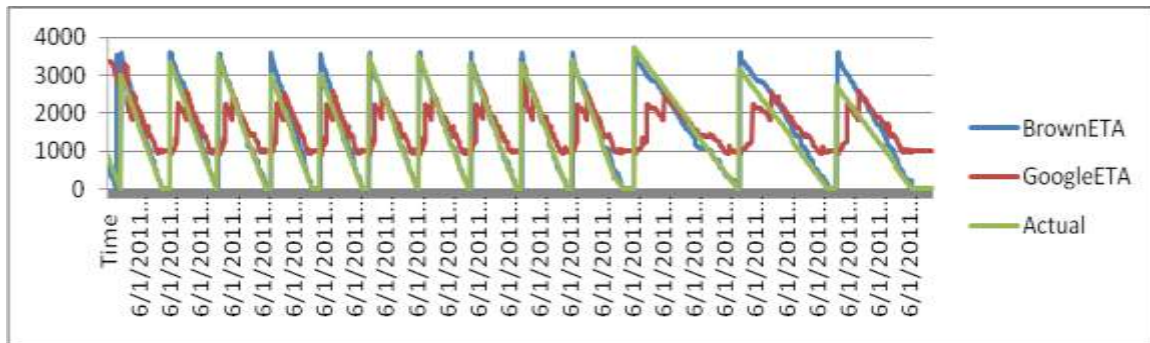


Figure 16: 104 East (Google/Brown)

As shown in figure 15 and 16, Dr. Brown's ETA turned out to be more accurate than the implementation of Google's direction service. The problem with Google's ETA is that there is a limitation of eight waypoints per direction request. With only eight waypoints the best way to keep a bus on its correct route was to use the buses stops as waypoints. Google will calculate the most efficient route to get to the next stop which will not necessarily match the buses actual route. Also, Google will not account for the bus stopping at the waypoints to pick up passengers which is a variable amount of time. The third challenge was the data in the AVL database was not normalized which made it difficult to pull out the right stops in the correct order. With more normalized data, adding an adjustment for stop times, and upgrading to Google Premier which allows for 25 waypoints per request, Google's ETA could be made much more accurate.

7. Summary

I was responsible for several deliverables that were attached to the completion of the project. Overall I was required to deliver a fully functional real time bus tracking data display system. This included several designs including an arrivals display, bus map display, and a mobile map interface. The design required focus on different aspects of each deliverable. For instance, the arrivals display was designed to be visually appealing to passengers at Forden Station while the mobile map interface was designed for simplicity and minimal load times.

Programming for the project helped to bring the designs to life by providing an interactive component as well as useful data. The most important aspect was to

communicate with the Google map API to bring Wave Transit's data to the map in an intuitive way. JavaScript played an important role not only in the Google Map API but to refresh elements on the screen without reloading the page. That combined with an XML document generated from a JSP file that communicated with the AVL database enabled the project requirements to be met. The programming deliverables included an interactive real time bus map and a real time arrival display that updates with current data every 30 seconds. The map also includes an information window that displays route, next stop, direction and ETA. A very simple administration page was provided for Wave Transit employees to quickly and easily update the news message on the arrivals display. Each deliverables is represented by several files including; HTML, CSS, JavaScript, JSP, Java and several images (see Appendix B).

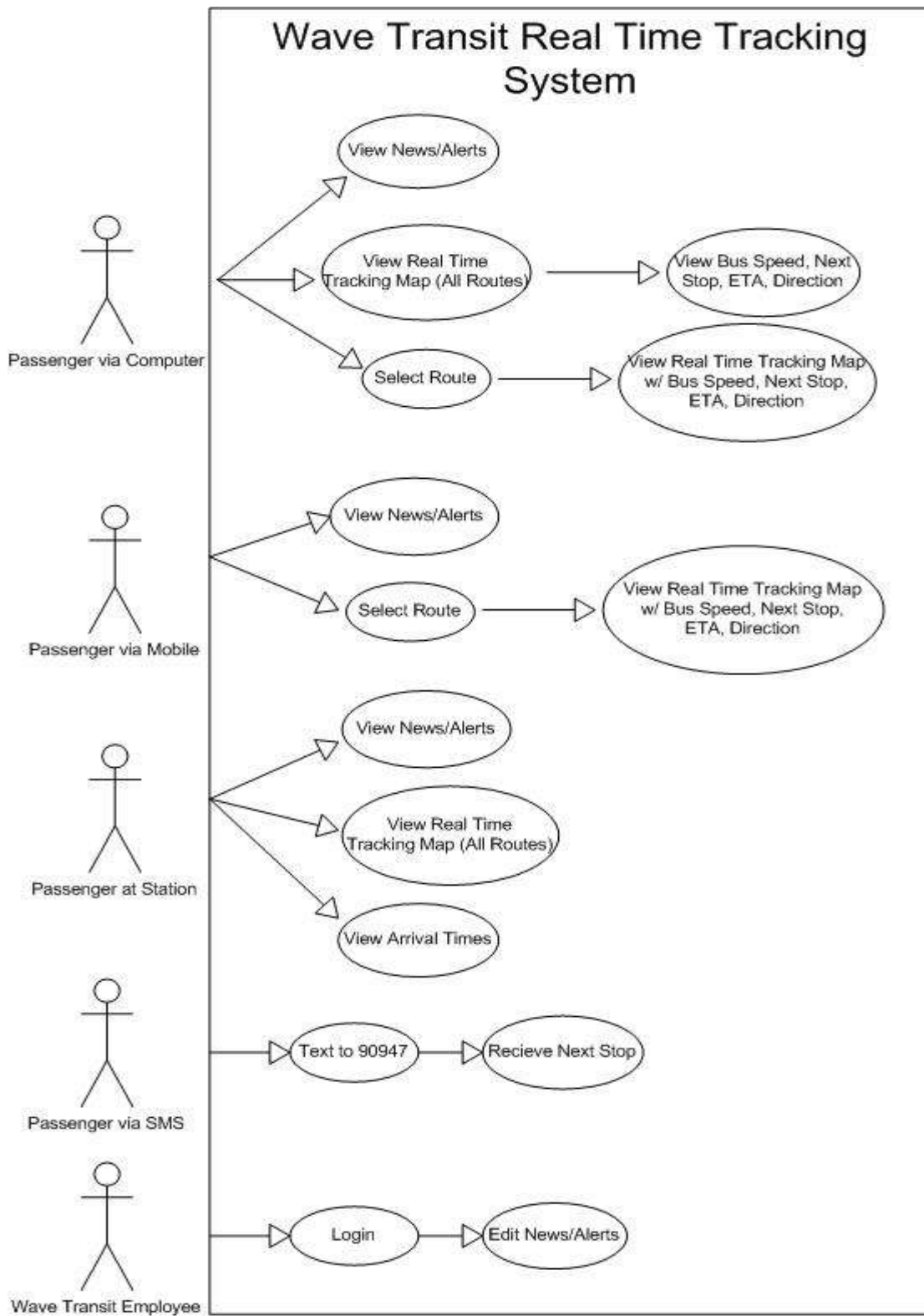
Finally, a simple analysis of Google ETA versus Dr. Jeff Brown's ETA was included to verify the accuracy of the provided estimations at Forden Station as well as the viability of using such methods. Google ETA was added to two different routes, 103 Central and 104 East. Charts were created to see if there were any patterns in the data to work with. From the charts it was easy to see that the implementation of Google's direction service would not be adequate to provide these estimations. Data was also collected on Dr. Brown's ETA for all six routes at the same time for an entire day. The charts resulting from this data collection show how accurate the estimations are. From the data that was collected Dr. Brown may adjust his algorithm to make it even more accurate in the future.

Wave Transit is very pleased with the new implementation of their real time bus tracking system. It exceeded their expectations in almost every way. The spiral model proved to be very beneficial throughout the project because the customer had suggestions for each prototype. It was very easy to add those suggestions in the requirements for the next prototype. Having the customer aware that updates could be accomplished so easily made the project go much more smoothly. With the system live and available to the public Wave Transit can now collect more suggestions through their online contact form and information desk at Forden Station for future enhancements.

8. Bibliography

1. "MTA BusTime." Metropolitan Transportation Authority, March 12, 2011.
<<http://bustime.mta.info/>>
2. "Extreme Programming." Wikipedia, The Free Encyclopedia. March 9, 2011.
<http://en.wikipedia.org/wiki/Extreme_Programming>
3. "Organizational Structure & Analysis." *Wave Transit - Wilmington, NC*. TJR Advisors, 12/2009. Web. 14 Mar 2011.
<http://www.wavetransit.com/org_study_final_report.pdf>.
4. "Spiral Model." Wikipedia, The Free Encyclopedia. March 8, 2011.
<http://en.wikipedia.org/wiki/Spiral_model>
5. "Online CS Modules: The Spiral Model." N.p., n.d. Web. 14 Mar 2011.
<<http://courses.cs.vt.edu/csonline/SE/Lessons/Spiral/index.html>>.
6. Melanson, Donald. "Brooklyn bus riders get real-time bus tracking via cellphone." *Engadget*. N.p., 05 02 2011. Web. 15 Mar 2011.
<<http://www.engadget.com/2011/02/05/brooklyn-bus-riders-get-real-time-bus-tracking-via-cellphone/>>.
7. "Town of Chapel Hill : Transit." Town of Chapel Hill. March 26, 2011.
<<http://www.townofchapelhill.org/index.aspx?page=1175>>
8. "Nextbus Homepage." Nextbus. March 26, 2011. <<http://www.nextbus.com/>>
9. "The NSCU Wolfline - Transit Visualization System". TransLoc. March 26, 2011.
<<http://ncsu.transloc.com/>>
10. ISO 8042 (1986) "Quality Vocabulary", International Organization for Standardization, Geneva
11. "CellSigns - Company :: Mobile Statistics". CellSigns. May 20, 2011.
<<http://www.cellsigns.com/industry.shtml>>
12. "Google Transit Partner Program". Google. May 26, 2011.
<<http://maps.google.com/help/maps/transit/partners/>>
13. Network Working Group of the IETF, January 2006, [RFC 4252](#), The Secure Shell (SSH) Authentication Protocol

Appendix A – Use Cases



Appendix B – Source Code for Web Page(s)

arrivals.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Central Station Departure Times</title>
<link rel="stylesheet" type="text/css" href="css/wavetransit.css" media="screen">
<script type="text/javascript" src="js/jquery-1.5.min.js"></script>
<script type="text/javascript">
    $(document).ready(function() {
        //initial data load
        loadData();
        loadMessage();
        //SetupClock
        updateClock();
        setInterval(updateClock,1000);
        //Load data every 30 seconds
        setInterval(loadData,30000);
        setInterval(loadMessage,30000);
    });
    function loadData(){
        $.ajax({
            url : "arrivals.jsp?iefix="+Math.floor(Math.random()*99999),
            success : function (data) {
                var temp = data.toString()
                var results = temp.split(",");
                $('#results td').each(function(index,domEle) {
                    if(index % 2 == 0)
                        $(domEle).html(results[index]);
                    else
                        if(results[index]<120)
                            $(domEle).html(Math.round(results[index]) + ' secs');
                        else
                            $(domEle).html(Math.round(results[index]/60) + ' mins');
                    if(index % 4 == 0 || (index-1) % 4 == 0)
                        $(domEle).css('background','#efefef');
                })
            }
        });
    }
    function loadMessage(){
        $.ajax({
            url : "arrivalsmessage.jsp?iefix="+Math.floor(Math.random()*99999),
            success : function (data) {
                var temp = data.toString()
                $("#message").html(temp);
            }
        });
    }
    function updateClock ( )
    {
        var currentTime = new Date ( );
```

```

var currentHours = currentTime.getHours ( );
var currentMinutes = currentTime.getMinutes ( );
var currentSeconds = currentTime.getSeconds ( );

// Pad the minutes and seconds with leading zeros, if required
currentMinutes = ( currentMinutes < 10 ? "0" : "" ) + currentMinutes;
currentSeconds = ( currentSeconds < 10 ? "0" : "" ) + currentSeconds;

// Choose either "AM" or "PM" as appropriate
var timeOfDay = ( currentHours < 12 ) ? "AM" : "PM";

// Convert the hours component to 12-hour format if needed
currentHours = ( currentHours > 12 ) ? currentHours - 12 : currentHours;

// Convert an hours component of "0" to "12"
currentHours = ( currentHours == 0 ) ? 12 : currentHours;

// Compose the string for display
var currentTimeString = currentHours + ":" + currentMinutes + " " + timeOfDay;

// Update the time display
$("#clock").html(currentTimeString);

        //Update date
        var month = currentTime.getMonth() + 1
        var day = currentTime.getDate()
        var year = currentTime.getFullYear()
        $("#date").html(month + "/" + day + "/" + year);

    }

</script>
</head>

<body>
<div id="header">
    <div id="clock"></div>
    <div id="date"></div>
</div>

<table width="100%" border="0" id="results" cellspacing="0">
    <tr>
        <td width="1010px">101 Brooklyn/Princess Place</td>
        <td></td>
    </tr>
    <tr>
        <td>102 University</td>
        <td></td>
    </tr>
    <tr>
        <td>103 Central</td>
        <td></td>
    </tr>
    <tr>
        <td></td>
        <td></td>
    </tr>

```

```

        <td>104 East</td>
        <td></td>
    </tr>
    <tr>
        <td>105 Medical center</td>
        <td></td>
    </tr>
    <tr>
        <td>106 West</td>
        <td></td>
    </tr>
</table>
<div id="message"></div>
</body>
</html>

```

arrivalsmessage.jsp

```

<% @ page language="java" import="java.sql.*"%>

<%
{
    Connection connection = null;
    try {
        String s = "SELECT messageoftheday from messageoftheday where id=1";
        Class.forName("net.sourceforge.jtds.jdbc.Driver");
        String s1 = "jdbc:jtds:sqlserver://test..com/test;user=test;password=test;";
        connection = DriverManager.getConnection(s1);
        ResultSet r = connection.createStatement().executeQuery(s);
        while(r.next()){
            out.println(r.getString("messageoftheday"));
        }
    }
    catch(Exception e){
        out.println(e.getMessage());
    }
    finally {
        try{connection.close();}
        catch(Exception ex){}
    }
}
%>

```

businfo.jsp

```

<% @ page language="java" import="java.sql.*"%>
<% @ page contentType="text/xml" %>

<buses>
<%
{
    Connection connection = null;
    try {
        String s = "SELECT VI.VehicleID AS vehicle_id, CONVERT(varchar, DATEADD(ss,
VI.UpdateTime, '01/01/1970'), 21) AS report_time, DATEDIFF(second,'1/1/1970', GETDATE())-
VI.UpdateTime AS Age, NULL as Address, CAST(VI.Latitude AS FLOAT)/100000 AS Lat,

```

```
CAST(VI.Longitude AS FLOAT)/100000 AS Lon, VI.Heading AS Dir, VI.Speed, VI.RouteName AS
line_id, VI.StopName AS NextStop, VI.OperatorName AS Driver FROM VehicleInfo VI WHERE
DATEDIFF(second,'1/1/1970',GETDATE())-VI.UpdateTime <= 90 AND VI.RouteID != '0' AND
VI.RouteID != '1000' order by 'line_id';
```

```
Class.forName("net.sourceforge.jtds.jdbc.Driver");
String s1 = "jdbc:jtds:sqlserver://test.com:1433/test;user=test;password=test;domain=test";
connection = DriverManager.getConnection(s1);
ResultSet r = connection.createStatement().executeQuery(s);
while(r.next()){
    out.println("<bus>");
    out.println("<vehicleId>" + r.getInt("vehicle_id") + "</vehicleId>");
    out.println("<reportTime>" + r.getString("report_time") + "</reportTime>");
    out.println("<latitude>" + r.getString("Lat") + "</latitude>");
    out.println("<longitude>" + r.getString("Lon") + "</longitude>");
    out.println("<direction>" + r.getString("Dir") + "</direction>");
    out.println("<speed>" + r.getString("Speed") + "</speed>");
    out.println("<lineId>" + r.getString("line_id") + "</lineId>");
    //out.println("<routeNumber>" + r.getString("RouteNum") + "</routeNumber>");
    out.println("<nextStop>" + r.getString("NextStop").replace("&","&amp;") + "</nextStop>");
    out.println("<driver>" + r.getString("Driver") + "</driver>");
    out.println("<age>" + r.getInt("age") + "</age>");
    out.println("</bus>");
}
}
catch(Exception e){
    out.println(e.getMessage());
}
finally {
    try{connection.close();}
    catch(Exception ex){}
}
}
%>

</buses>
```

stationmap.html

```
<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
<title>Wave Transit Real Time Tracking</title>
<style type="text/css">
html { height: 100% }
body { height: 100%; margin: 0px; padding: 0px }
#map_canvas { height: 100% }
.labels {
    color: black;
    background-color: white;
    font-family: "Lucida Grande", "Arial", sans-serif;
    font-size: 12px;
    font-weight: bold;
    text-align: center;
    border: 2px solid black;
    white-space: nowrap;
```

```

    }
</style>

<script type="text/javascript" src="js/jquery-1.5.min.js"></script>

<script type="text/javascript"
src="http://maps.google.com/maps/api/js?libraries=geometry&sensor=false"></script>
<script type="text/javascript" src="js/markerwithlabel.js"></script>

<script type="text/javascript">
    var markersArray = [];
    var zoomlevel = 14;
    var centerchanged = false;

    //On Body Load initialize map and load XML
    function initialize() {
        var latlng = new google.maps.LatLng(34.22394196192143, -77.87800683593754);
        var myOptions = {
            zoom: zoomlevel,
            center: latlng,
            mapTypeId: google.maps.MapTypeId.ROADMAP
        };

        map = new google.maps.Map(document.getElementById("map_canvas"),myOptions);

        //Create a you are here icon
        var image = new google.maps.MarkerImage('images/youarehere.png',
            // This marker is 20 pixels wide by 32 pixels tall.
            new google.maps.Size(115, 100),
            // The origin for this image is 0,0.
            new google.maps.Point(0,0),
            // The anchor for this image is the base of the flagpole at 0,32.
            new google.maps.Point(0, 100)
        );

        //Mark the station with you are here icon
        var myLatLng = new google.maps.LatLng(34.251127,-77.874952);
        var stationmarker = new google.maps.Marker({
            position: myLatLng,
            map: map,
            icon: image,
            title: 'Forden Station',
            zIndex: 10
        });

        //initial xml load
        loadXML();
        //Load XML every 30 seconds
        setInterval(loadXML,30000);
    }
    //makes AJAX request to load xml data from businfo.jsp
    function loadXML(){
        $.ajax({
            type: "GET",

```

```

        url: "businfo.jsp?iefix="+Math.floor(Math.random()*99999),
        dataType: "xml",
        success: parseXml
    });
}
//on XML load remove overlay and recreate current overlays
function parseXml(xml) {
    //remove current buses
    deleteOverlays()
    //find every bus and mark it
    $(xml).find("bus").each(function()
    {
        //create marker
        var marker = new MarkerWithLabel({
            position: new
            google.maps.LatLng($(this).find("latitude").text(),$(this).find("longitude").text()),
            map: map,
            title: $(this).find("lineId").text(),
            labelContent: $(this).find("lineId").text(),
            labelAnchor: new google.maps.Point(50,30),
            labelClass: "labels", // the CSS class for the label
            labelStyle: { zIndex: Math.floor(Math.random()*30)}
        });

        markersArray.push(marker);
    });
}
// Deletes all markers in the array by removing references to them
function deleteOverlays() {
    if (markersArray) {
        for (i in markersArray) {
            markersArray[i].setMap(null);
        }
        markersArray.length = 0;
    }
}
</script>

</head>

<body onLoad="initialize()">
    <div id="map_canvas" style="width:100%; height:100%"></div>
</body>
</html>

```

arrivals.jsp

```

<% @ page contentType="text/html; charset=utf-8" language="java"
import="java.net.*,java.io.*,java.util.*;" %>
<%

try {
    String retVal = "";
    URL u;
    HttpURLConnection uc;

    u = new URL("http://mymobed.com/map/all.jsp");

```



```

uc = (URLConnection)u.openConnection();

InputStream content = (InputStream)uc.getInputStream();
BufferedReader inb = new BufferedReader (new InputStreamReader(content));
String line;
while ((line = inb.readLine()) != null) {
    retVal += line;
}
out.println(retVal);
}
catch(Exception e){
    e.printStackTrace();
}
%>

```

map.html

```

<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
<title>Wave Transit Real Time Tracking</title>
<link rel="stylesheet" type="text/css" href="css/style.css" media="screen">
<link rel="stylesheet" href="css/mobile.css" media="handheld, screen and (max-device-width: 480px)"
type="text/css" />
<style type="text/css">
    html { }
    body { margin: 0px; padding: 0px;background:transparent; }
    .labels {
        color: black;
        background-color: white;
        font-family: "Lucida Grande", "Arial", sans-serif;
        font-size: 12px;
        font-weight: bold;
        text-align: center;
        border: 2px solid black;
        white-space: nowrap;
    }
</style>

<script type="text/javascript" src="js/jquery-1.5.min.js"></script>

<script type="text/javascript"
src="http://maps.google.com/maps/api/js?libraries=geometry&sensor=false"></script>
<script type="text/javascript" src="js/markerwithlabel.js"></script>

<script type="text/javascript">
    var markersArray = [];
    var directionsService = new google.maps.DirectionsService();
    //var directionsDisplay = new google.maps.DirectionsRenderer();
    var refine = "";
    var infowindow = new google.maps.InfoWindow({
        size: new google.maps.Size(150,50),
        disableAutoPan:true
    });

    var zoomlevel = 14;

```

```

var centerchanged = false;

//On Body Load initialize map and load XML
function initialize() {
    var latlng = new google.maps.LatLng(34.22408389926041, -77.90427102661137);
    var myOptions = {
        zoom: zoomlevel,
        mapTypeId: google.maps.MapTypeId.ROADMAP
    };
    map = new google.maps.Map(document.getElementById("map_canvas"),myOptions);
    //directionsDisplay.setMap(map);

    //initial xml load
    loadXML();
    //Load XML every 30 seconds
    setInterval(loadXML,30000);
    //check if the user is controlling the map...if so don't move it
    google.maps.event.addListener(map, 'center_changed', function() {
        centerchanged=true;
    });
    //when the user closes the info window deselect the bus
    google.maps.event.addListener(infowindow, 'closeclick', function() {
        if(refine != "")
            window.location.hash = refine;
        else
            window.location.hash = "";
    });
}
function calcRoute(start,end) {

}
//makes AJAX request to load xml data from businfo.jsp
function loadXML(){
    if(window.location.hash.length > 4){
        refine=window.location.hash;
        refine=refine.replace("#","");
    }

    $.ajax({
        type: "GET",
        url:
        "businfo.jsp?refine="+refine+"&iefix="+Math.floor(Math.random()*99999),
        dataType: "xml",
        success: parseXml
    });
}
//on XML load remove overlay and recreate current overlays
function parseXml(xml) {
    //remove current buses
    deleteOverlays()
    //find every bus and mark it
    $(xml).find("bus").each(function()
    {

        var latitude = $(this).find("latitude").text();
        var longitude = $(this).find("longitude").text();
        var nextlatitude = $(this).find("nextLatitude").text();
    }

```

```

var nextlongitude = $(this).find("nextLongitude").text();
//create marker
var marker = new MarkerWithLabel({
    position: new google.maps.LatLng(latitude,longitude),
    map: map,
    title: $(this).find("lineId").text(),
    labelContent: $(this).find("lineId").text(),
    labelAnchor: new google.maps.Point(50,30),
    labelClass: "labels" // the CSS class for the label
});
marker.set("id", $(this).find("vehicleId").text());

//create content for info window
var contentString = "<strong>" + $(this).find("lineId").text() + "</strong><br>Next Stop: " + $(this).find("nextStop").text() + "<br>Speed: " + $(this).find("speed").text() + "mph" + "<br>Direction: " + headingToString($(this).find("direction").text());

//add event listener to show info window when user clicks marker
google.maps.event.addListener(marker, 'click', function() {
    start = new google.maps.LatLng(latitude,longitude);
    end = new google.maps.LatLng(nextlatitude,nextlongitude);
    var request = {
        origin:start,
        destination:end,
        travelMode: google.maps.TravelMode.DRIVING
    };
    directionsService.route(request, function(result, status) {
        if (status == google.maps.DirectionsStatus.OK) {
            //directionsDisplay.setDirections(result);
            duration = result.routes[0].legs[0].duration.value;
            if(duration <120)
                formatted = duration + ' secs';
            else
                formatted = Math.round(duration/60) + ' mins';

            infowindow.setContent(contentString+'<br>ETA: '+formatted );

            infowindow.open(map,marker);
            //select and follow the bus if user clicks the marker
            window.location.hash = "#" + marker.get("id");
        }
        else {
            infowindow.setContent(contentString );
            infowindow.open(map,marker);
            //select and follow the bus if user clicks the marker
            window.location.hash = "#" + marker.get("id");
        }
    });
});

//zoom in on bus and display info window if selected
if(window.location.hash == "#" + $(this).find("vehicleId").text()){
    //if a vehicle is selected center the map on it else user has control dont move it
    if(!centerchanged){

```

```

        map.panTo(new
        google.maps.LatLng($(this).find("latitude").text(),$(this).find("longitude").text()));
        centerchanged=false;
    }
    //if a vehicle has been selected always show the info window
    google.maps.event.trigger(marker, 'click');
}

        markersArray.push(marker);
    });
    //if(window.location.hash.length > 4){
    if(!centerchanged){
        var bound = new google.maps.LatLngBounds();
        for(var i in markersArray)
        {
            bound.extend(markersArray[i].getPosition());
        }
        map.fitBounds(bound);
        centerchanged=false;
        if (markersArray.length==1)
            map.setZoom(15);
    }
    //}
}
// Deletes all markers in the array by removing references to them
function deleteOverlays() {
    if (markersArray) {
        for (i in markersArray) {
            markersArray[i].setMap(null);
        }
        markersArray.length = 0;
    }
}
function headingToString(x)
{
    var directions = new Array("N", "NE", "E", "SE", "S", "SW", "W", "NW", "N");
    direction = Math.round(( x % 360) / 45);
    return directions[direction];
}
}
</script>

</head>

<body onLoad="initialize()">
    <ul class="nav" id="goback">
        <li>
            <a href='http://coupons2yourphone.com/bustracking/wavetransit/m/select-
            route.jsp?method=map'>< Go Back</a>
        </li>

    </ul>
    <div id="map_canvas"></div>
</body>
</html>

```

index.jsp

```
<% @ page contentType="text/html; charset=UTF-8" language="java" import="java.sql.*" errorPage=""
%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<meta name="viewport" content="width=device-width" />
<title>Select a Route</title>
<link rel="stylesheet" type="text/css" href="css/style.css" media="screen">
<link rel="stylesheet" href="css/mobile.css" media="handheld, screen and (max-device-width: 480px)"
type="text/css" />

</head>

<body>
<div class="content">
  <ul class="nav">
    <li>
      <a href='select-route.jsp?method=table'>View Bus Route/ETA Table</a>
    </li>
    <li>
      <a href='select-route.jsp?method=map'>View Single Bus Route Map</a>
    </li>
    <li class="allbus">
      <a href='map.html'>View All Bus Route Map</a>
    </li>
  </ul>
</div>
</body>
</html>
```

select-route.jsp

```
<% @ page contentType="text/html; charset=UTF-8" language="java" import="java.sql.*" errorPage=""
%>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" lang="en">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<meta name="viewport" content="width=device-width" />
<title>Select a Route</title>
<link rel="stylesheet" type="text/css" href="css/style.css" media="screen">
<link rel="stylesheet" href="css/mobile.css" media="handheld, screen and (max-device-width: 480px)"
type="text/css" />

</head>

<body>
<!--<div class="header">
  <p> <strong>Select a Route</strong></p>
</div-->
<div class="content">
```

```

<ul class="nav">
<%
{
    String method = request.getParameter("method") ;
    Connection connection = null;
    try {
        int num = 0;
        String s = "SELECT Distinct VI.RouteName AS line_id FROM VehicleInfo VI WHERE
DATEDIFF(second,'1/1/1970',GETDATE())-VI.UpdateTime <= 90 AND VI.RouteName<>'OUT OF
SERVICE' AND VI.RouteID != '0' AND VI.Latitude>0.0 AND VI.RouteID != '1000' order by 'line_id' ";
        Class.forName("net.sourceforge.jtds.jdbc.Driver");
        String s1 = "jdbc:jtds:sqlserver://test.com:1433/test;user=test;password=test;domain=test";
        connection = DriverManager.getConnection(s1);
        ResultSet r = connection.createStatement().executeQuery(s);
        while(r.next()){
            out.println("<li>");
            out.println("<a href='"+method+".html#"+r.getString("line_id").replace(" ","%20")+"'>" +
r.getString("line_id")+ "</a>");
            out.println("</li>");
            num++;
        }
        if(num==0)
            out.println("<i>There are no active vehicles at this time</i>");
    }
    catch(Exception e){
        out.println(e.getMessage());
    }
    finally {
        try{connection.close();}
        catch(Exception ex){}
    }
}
%>
</ul>
</div>
</body>
</html>

```

table.html

```

<!DOCTYPE html>
<html>
<head>
<meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
<title>Wave Transit Real Time Tracking</title>
<link rel="stylesheet" type="text/css" href="css/mobile.css" media="screen">

<script type="text/javascript" src="js/jquery-1.5.min.js"></script>
<script type="text/javascript"
src="http://maps.google.com/maps/api/js?libraries=geometry&sensor=false"></script>

<script type="text/javascript">
    var directionsService = new google.maps.DirectionsService();
    var refine = "";

```

```

//On Body Load initialize map and load XML
function initialize() {
    //initial xml load
    loadXML();
    //Load XML every 30 seconds
    setInterval(loadXML,30000);
}

//makes AJAX request to load xml data from businfo.jsp
function loadXML(){
    if(window.location.hash.length > 4){
        refine=window.location.hash;
        refine=refine.replace("#","");
    }

    $.ajax({
        type: "GET",
        url:
"businfo.jsp?refine="+refine+"&iefix="+Math.floor(Math.random()*99999),
        dataType: "xml",
        success: parseXml
    });
}

//on XML load remove overlay and recreate current overlays
function parseXml(xml) {
    $("ul li").remove();
    //find every bus and mark it
    $(xml).find("bus").each(function()
    {

        var latitude = $(this).find("latitude").text();
        var longitude = $(this).find("longitude").text();
        var nextlatitude = $(this).find("nextLatitude").text();
        var nextlongitude = $(this).find("nextLongitude").text();
        var lineid = $(this).find("lineId").text();

        //create content for info window
        var contentString = "<strong>"+$(this).find("lineId").text()+"</strong><br/>Next Stop:
"+$(this).find("nextStop").text()+"<br/>Speed: "+$(this).find("speed").text()+"mph"+<br/>Direction:
"+headingToString($(this).find("direction").text());

        start = new google.maps.LatLng(latitude,longitude);
        end = new google.maps.LatLng(nextlatitude,nextlongitude);
        var request = {
            origin:start,
            destination:end,
            travelMode: google.maps.TravelMode.DRIVING
        };
        directionsService.route(request, function(result, status) {
            if (status == google.maps.DirectionsStatus.OK) {
                //directionsDisplay.setDirections(result);
                duration = result.routes[0].legs[0].duration.value;
                if(duration <120)
                    formatted = duration + ' secs';
                else
                    formatted = Math.round(duration/60) + ' mins';
            }
        });
    });
}

```

```

        $(".nav").append("<li>" + contentString + "<br/>ETA: " + formatted + "</li>");
    }
    else {
        $(".nav").append("<li>" + contentString + "</li>");
    }
});

});
}
// Deletes all markers in the array by removing references to them
function headingToString(x)
{
    var directions = new Array("N", "NE", "E", "SE", "S", "SW", "W", "NW", "N");
    direction = Math.round(( x % 360) / 45);
    return directions[direction];
}
}
</script>

</head>
<body onLoad="initialize()">
    <div class="content">
        <ul class="nav">
        </ul>
    </div>
</body>
</html>

```

mobile.css

```

@charset "UTF-8";
/* CSS Document */
html{height: 100%;}
body{
    color: #515151;
    font-family: Arial,Helvetica,sans-serif;
    margin: 0 auto;
    width:100%;
    background:transparent;
    height: 100%;
}
#map_canvas { width:100%; height:100% }
.labels {
    color: black;
    background-color: white;
    font-family: "Lucida Grande", "Arial", sans-serif;
    font-size: 12px;
    font-weight: bold;
    text-align: center;
    border: 2px solid black;
    white-space: nowrap;
}
p{
    margin:0;
    padding:0;
}

```



```

}
a{
    color: #515151;
    text-decoration:none;
}
a:hover,a:visited{
    color: #515151;
    text-decoration:underline;
}
h1{
    color:white;
    font-size: 24px;
}
.header{
    height:36px;
    background: url(images/mobile-header-bg.gif) repeat-x top left;
    color:white;
}

ul {
    border: 1px solid #D9D9D9;
}
ul {
    list-style: none;
    padding: 0px;
}
li {
    border-top: 1px solid #D9D9D9;
}
li:first-child {
    border: none;
}
li a, .list_items li {
    display: block;
    padding: 10px;
}
li a:hover {
    background-color: #f3f3f3;
}
.content{
    margin:-11px 5px 5px 5px;
}
.nav {
    background-color: #efefef;
    margin-top: 10px;
    margin-bottom: 10px;
    margin-left: 0px;
}
.nav a:hover {
    background-color: #b2daff;
}
#goback{
    display:none;
}
.allbus{
    display:none;
}

```

```
}
```

style.css

```
@charset "UTF-8";
/* CSS Document */
body{
    color: #515151;
    font-family: Arial,Helvetica,sans-serif;
    margin: 0 auto;
    width:100%;
    background:transparent;
}
#map_canvas { height:380px; }
.labels {
    color: black;
    background-color: white;
    font-family: "Lucida Grande", "Arial", sans-serif;
    font-size: 12px;
    font-weight: bold;
    text-align: center;
    border: 2px solid black;
    white-space: nowrap;
}
p{
    margin:0;
    padding:0;
}
a{
    color: #515151;
    text-decoration:none;
}
a:hover,a:visited{
    color: #515151;
    text-decoration:underline;
}
h1{
    color:white;
    font-size: 24px;
}
.header{
    height:36px;
    background: url(images/mobile-header-bg.gif) repeat-x top left;
    color:white;
}
ul {
    border: 1px solid #D9D9D9;
}
ul {
    list-style: none;
    padding: 0px;
}
li {
    border-top: 1px solid #D9D9D9;
}
li:first-child {
```

```

        border: none;
    }
    li a, .list_items li {
        display: block;
        padding: 10px;
    }
    li a:hover {
        background-color: #f3f3f3;
    }
    .content{
        margin:-11px 5px 5px 5px;
    }
    .nav {
        background-color: #efefef;
        margin-top: 10px;
        margin-bottom: 10px;
        margin-left: 0px;
    }
    .nav a:hover {
        background-color: #b2daff;
    }

```

Appendix C – Source Code for Data Display LEDs

WaveTransitCRON.sh

```
#
# WaveTransitCRON.sh - Script for the Wave Transit LED Data Display System
# Last Update: May 10, 2011
#
# Create cron job (specified in 'crontab_file') to run this script every minute of every day
# "*" * * * * sh /home/hooks/WaveTransitLED/WaveTransitCRON.sh"
#
# To install cron job for user 'hooks'
# "crontab -u hooks crontab_file"
#
# To list cron job for user 'hooks'
# "crontab -u hooks -l"
#
# To remove cron job for user 'hooks'
# "crontab -u hooks -r"
#
# Run twice in one minute
cd /home/hooks/WaveTransitLED
/home/hooks/jdk1.6.0_24/bin/java WaveTransitLEDControllerOneTime
sleep 30
/home/hooks/jdk1.6.0_24/bin/java WaveTransitLEDControllerOneTime
exit 0
```

crontab_file

```
* * * * * sh /home/hooks/WaveTransitLED/WaveTransitCRON.sh
```

WaveTransitCRON.sh

```
#
# WaveTransitCRON.sh - Script for the Wave Transit LED Data Display System
#
# Last Update: May 10, 2011
#
# Create cron job (specified in 'crontab_file') to run this script every minute of every day
# "*" * * * * sh /home/hooks/WaveTransitLED/WaveTransitCRON.sh"
#
# To install cron job for user 'hooks'
# "crontab -u hooks crontab_file"
#
# To list cron job for user 'hooks'
# "crontab -u hooks -l"
#
# To remove cron job for user 'hooks'
# "crontab -u hooks -r"
#
# Run twice in one minute
cd /home/hooks/WaveTransitLED
/home/hooks/jdk1.6.0_24/bin/java WaveTransitLEDControllerOneTime
sleep 30
/home/hooks/jdk1.6.0_24/bin/java WaveTransitLEDControllerOneTime
exit 0
```

menu2.sh

```
#
# Script for the Wave Transit LED Data Display System
#
# Last Update: May 10, 2011
#
while [ 1 ]
do
    clear
    echo "Welcome to the Wave Transit LED Data Display System"
    echo
    echo "1. Stop LED Data Display Program (do this before choosing item 2 or 3 below)"
    echo "2. Start LED Data Display Program (LED program will be left running)"
    echo "3. Execute Additional Menu Options (send commands to individual signs)"
    echo "4. Exit Program"
    echo
    echo "Enter your selection:"
    read selection
    case $selection in
        "1") crontab -u hooks -r
            echo
            echo "It will take the LED Controller Program 30 Seconds to Stop"
            echo
            echo "Please Wait....."
            echo
            sleep 30 # wait worse case for cron to stop shell program
            # Now run program to clear all displays
            java WaveTransitLEDControllerClearAll
            echo "All displays have been cleared"
            echo "Press any key to continue"
            read keyboard;;
        "2") crontab -u hooks crontab_file
            echo
            echo "LED Controller Program Started in the Background as a Cron Job"
            echo
            echo "It may take up to 1 minute for the displays to be updated"
            echo
            echo "Press any key to continue"
            read keyboard;;
        "3") java WaveTransitLEDMenu
            echo "Press any key to continue"
            read keyboard;;
        "4") break;;
        *) echo "Invalid selection $selection - try again";;
    esac
done
echo
echo "Please type 'exit' to close the terminal window"
echo
exit 0
```

WaveTransitLEDcontrollerClearAll.java

```
import java.io.*;
import java.net.*;
import java.util.*;
public class WaveTransitLEDControllerClearAll
{
    public static void main(String[] args) throws InterruptedException
    {
        final int ACTIVE_DISPLAYS = 10; // ten displays
        final int MAX_DISPLAYS = 10;    // max of ten bays at some point
        String message, command;

        // Setup sockets and streams
        Socket[] ledSocket = new Socket[MAX_DISPLAYS]; // client sockets - one per display
        DataOutputStream[] os = new DataOutputStream[MAX_DISPLAYS]; // output streams - one per
display
        DataInputStream[] is = new DataInputStream[MAX_DISPLAYS]; // input streams - one per display

        String[] IP = new String[MAX_DISPLAYS]; // IP number of each display
        IP[0] = "192.168.41.80"; IP[1] = "192.168.41.81";
        IP[2] = "192.168.41.82"; IP[3] = "192.168.41.83";
        IP[4] = "192.168.41.84"; IP[5] = "192.168.41.85";
        IP[6] = "192.168.41.86"; IP[7] = "192.168.41.87";
        IP[8] = "192.168.41.88"; IP[9] = "192.168.41.89";

        // Try to open sockets on port 10002 and all input and output streams
        try {
            for (int i=0; i<ACTIVE_DISPLAYS; i++)
            {
                ledSocket[i] = new Socket(IP[i], 10002);
                os[i] = new DataOutputStream(ledSocket[i].getOutputStream());
                is[i] = new DataInputStream(ledSocket[i].getInputStream());
            }
        }
        catch (UnknownHostException e) {
            System.err.println("Don't know what host this is.");
            System.exit(-1);
        }
        catch (IOException e) {
            System.err.println("Couldn't get I/O for the connection.");
            System.exit(-1);
        }

        // clear all displays
        for (int i=0; i<ACTIVE_DISPLAYS; i++)
        {
            if (ledSocket[i] != null && os[i] != null && is[i] != null)
            {
                try {
                    command = "!pr"; // delete all pages and clear the display
                    message = "[0@" + command + "]; // send two nulls: '0' and '@' so no reply is returned
                    os[i].writeBytes(message);
                    //System.out.println("Deleted pages and cleared displays");
                }
                catch (UnknownHostException e) {
                    System.err.println("Trying to connect to unknown host: " + e);
                }
            }
        }
    }
}
```

```

    }
    catch (IOException e) {
        System.err.println("IOException: " + e);
    }
}
}

// clean up: close input streams, output streams and sockets
for (int i=0; i<ACTIVE_DISPLAYS; i++)
{
    try {
        os[i].close();
        is[i].close();
        ledSocket[i].close();
    }
    catch (UnknownHostException e) {
        System.err.println("Trying to connect to unknown host: " + e);
    }
    catch (IOException e) {
        System.err.println("IOException: " + e);
    }
}
}
}

```

WaveTransitLEDControllerOneTime.java

```

//
// WaveTransitLEDControllerOneTime.java - opens sockets to all displays, put data on them, then closes
// all sockets
//
// Last Updated: May 10, 2011
//
import java.io.*;
import java.net.*;
import java.util.*;
import java.util.Calendar;
import java.text.SimpleDateFormat;
public class WaveTransitLEDControllerOneTime
{
    public static void main(String[] args) throws InterruptedException
    {
        final int START_TIME = 0600; // 6AM
        final int END_TIME = 2200; // 9PM
        final int ACTIVE_DISPLAYS = 10; // ten displays
        final int MAX_DISPLAYS = 10; // max of ten bays at some point
        String message, command;

        // Setup sockets and streams
        Socket[] ledSocket = new Socket[MAX_DISPLAYS]; // client sockets - one per display
        DataOutputStream[] os = new DataOutputStream[MAX_DISPLAYS]; // output streams - one per
display
        DataInputStream[] is = new DataInputStream[MAX_DISPLAYS]; // input streams - one per display

        String[] IP = new String[MAX_DISPLAYS]; // IP number of each display
        IP[0] = "192.168.41.80"; IP[1] = "192.168.41.81";
    }
}

```

```

IP[2] = "192.168.41.82"; IP[3] = "192.168.41.83";
IP[4] = "192.168.41.84"; IP[5] = "192.168.41.85";
IP[6] = "192.168.41.86"; IP[7] = "192.168.41.87";
IP[8] = "192.168.41.88"; IP[9] = "192.168.41.89";

String [] LINE1 = new String[MAX_DISPLAYS]; // Line 1 of ETA
String [] LINE2 = new String[MAX_DISPLAYS]; // Line 2 of ETA
String [] DISPLAY1 = new String[MAX_DISPLAYS]; // Line 1 of Display
String [] DISPLAY2 = new String[MAX_DISPLAYS]; // Line 2 of Display

for (int i=0; i<ACTIVE_DISPLAYS; i++)
{
    DISPLAY1[i] = "Welcome to";
    DISPLAY2[i] = "Wave Transit";
}

// Try to open sockets on port 10002 and all input and output streams
try {
    for (int i=0; i<ACTIVE_DISPLAYS; i++)
    {
        ledSocket[i] = new Socket(IP[i], 10002);
        os[i] = new DataOutputStream(ledSocket[i].getOutputStream());
        is[i] = new DataInputStream(ledSocket[i].getInputStream());
    }
}
catch (UnknownHostException e) {
    System.err.println("Don't know what host this is.");
    System.exit(-1);
}
catch (IOException e) {
    System.err.println("Couldn't get I/O for the connection.");
    System.exit(-1);
}

// do one time
Calendar cal = Calendar.getInstance();
SimpleDateFormat sdf = new SimpleDateFormat("Hmm");
String systime = sdf.format(cal.getTime());
int current_systime = Integer.parseInt(systime);

//System.out.println("30 second loop - time is: " + systime);

if ((current_systime > START_TIME) && (current_systime < END_TIME))
{
    // Get ETA data and put into line1 and line2
    getETAvalues(LINE1, LINE2);

    // Assign to DISPLAYS
    DISPLAY1[0] = "Welcome to"; // 192.168.41.80 -> no route
    DISPLAY2[0] = "Wave Transit";

    DISPLAY1[1] = LINE1[5]; // 192.168.41.81 -> Route 106
    DISPLAY2[1] = LINE2[5];

    DISPLAY1[2] = LINE1[4]; // 192.168.41.82 -> Route 105
    DISPLAY2[2] = LINE2[4];
}

```



```

DISPLAY1[3] = LINE1[1];    // 192.168.41.83 -> Route 102
DISPLAY2[3] = LINE2[1];

DISPLAY1[4] = LINE1[0];    // 192.168.41.84 -> Route 101
DISPLAY2[4] = LINE2[0];

DISPLAY1[5] = "Welcome to"; // 192.168.41.85
DISPLAY2[5] = "Wave Transit";

DISPLAY1[6] = "Welcome to"; // 192.168.41.86
DISPLAY2[6] = "Wave Transit";

DISPLAY1[7] = "Welcome to"; // 192.168.41.87 -> Teal
DISPLAY2[7] = "Wave Transit";

DISPLAY1[8] = LINE1[3];    // 192.168.41.88 -> Route 104
DISPLAY2[8] = LINE2[3];

DISPLAY1[9] = LINE1[2];    // 192.168.41.89 -> Route 103
DISPLAY2[9] = LINE2[2];

//display routes and ETAs on all active displays
for (int i=0; i<ACTIVE_DISPLAYS; i++)
{
    if (ledSocket[i] != null && os[i] != null && is[i] != null)
    {
        try {
            //construct display lines
            command = "!ps01!m1!c11"; // Page store 01; mode 1; clear line 1
            command = command + "!dl1jpL"; // Display line 1; jump w/text appears instantly; left
justify
            command = command + "<\n" + DISPLAY1[i] + ">";
            command = command + "!c12"; // Clear line 2
            command = command + "!dl2jpL"; // Display line 2; jump w/text appears instantly; left
justify;
            command = command + "<\n" + DISPLAY2[i] + ">";
            command = command + "!dp01!pl01"; // Display pause 001 second; set page list order to
page 1
            message = "[0@" + command + "]"; // send two nulls: '0' and '@' so no reply is returned
            os[i].writeBytes(message);
        }
        catch (UnknownHostException e) {
            System.err.println("Trying to connect to unknown host: " + e);
        }
        catch (IOException e) {
            System.err.println("IOException: " + e);
        }
    }
}
}
else
{ // outside of display start and end times
    try {
        for (int i=0; i<ACTIVE_DISPLAYS; i++)
        {

```

```

        command = "!pr"; // delete all pages and clear the display
        message = "[0@" + command + "]; // send two nulls: '0' and '@' so no reply is returned
        os[i].writeBytes(message);
    }
    //System.out.println("Deleted pages and cleared displays");
}
catch (UnknownHostException e) {
    System.err.println("Trying to connect to unknown host: " + e);
}
catch (IOException e) {
    System.err.println("IOException: " + e);
}
}

// clean up: close input streams, output streams and sockets
for (int i=0; i<ACTIVE_DISPLAYS; i++)
{
    try {
        os[i].close();
        is[i].close();
        ledSocket[i].close();
    }
    catch (UnknownHostException e) {
        System.err.println("Trying to connect to unknown host: " + e);
    }
    catch (IOException e) {
        System.err.println("IOException: " + e);
    }
}

}

// Get ETA values from myMobEd.com server
public static void getETAvalues(String [] LINE1, String [] LINE2)
{
    String line, retVal = "";
    URL u;
    HttpURLConnection uc;

    try {
        u = new URL("http://mymobed.com/map/all.jsp");
        uc = (HttpURLConnection)u.openConnection();

        InputStream content = (InputStream)uc.getInputStream();
        BufferedReader inb = new BufferedReader (new InputStreamReader(content));

        while ((line = inb.readLine()) != null) {
            retVal += line;
        }
        //System.out.println(retVal + "\n");

        StringTokenizer stk = new StringTokenizer(retVal, ",", false);
        int i = 0;
        while(stk.hasMoreTokens()){
            // System.out.println("Route: "+stk.nextToken()+" ETA: "+stk.nextToken());
            LINE1[i] = "Route: " + stk.nextToken();

```

```

        String strVal = stk.nextToken();
        int ETAtime = Integer.parseInt(strVal);
        if (ETAtime < 120) // 60 seconds
            LINE2[i] = "ETA: " + strVal + " seconds";
        else
            LINE2[i] = "ETA: " + Integer.toString(ETAtime/60) + " minutes";

        i++;
    }
}
catch(Exception e){
    e.printStackTrace();
}

}
}

```

WaveTransitLEDMenu.java

```

import java.io.*;
import java.net.*;
import java.util.Scanner;
public class WaveTransitLEDMenu
{
    public static void main(String[] args) {
        Socket ledSocket = null;
        DataOutputStream os = null; // output stream
        DataInputStream is = null; // input stream
        Scanner keyboard = new Scanner(System.in);

        // Choose a display
        System.out.print("\n\nSelect an active display 192.168.41.xx (where xx = 80 to 89): ");
        String input = keyboard.nextLine();
        String IP = "192.168.41." + input;

        // Try to open input and output streams on port 10002
        try {
            ledSocket = new Socket(IP, 10002);
            os = new DataOutputStream(ledSocket.getOutputStream());
            is = new DataInputStream(ledSocket.getInputStream());
        } catch (UnknownHostException e) {
            System.err.println("Unknown Host Exception " + IP);
        } catch (IOException e) {
            System.err.println("IO Exception for " + IP);
        }
        // If everything has been initialized then we want to write some data
        // to the socket we have opened a connection to on port 10002
        if (ledSocket != null && os != null && is != null) {
            try {

                while (true)
                {
                    System.out.println("\n --- Main Menu ---");
                    System.out.print(" 1. Set RTC Time\n 2. Set RTC Date\n 3. Set Display Brightness\n");
                    System.out.print(" 4. Get HW Serial Number\n 5. Get Firmware ID and Version\n");
                    System.out.print(" 6. Get HW Status\n 7. Get Display Configuration\n");
                }
            }
        }
    }
}

```

```

System.out.print(" 8. Get Time\n 9. Get Date\n");
System.out.print("10. Get Operational Status\n11. Get Error Status\n");
System.out.print("12. LED Test\n13. Get # of Failed LEDs\n");
System.out.print("14. Display Sample Text\n15. Execute Poll Command\n");
System.out.print("16. Delete Pages and Clear Display\n17. Reboot Display\n");
System.out.print("18. Exit and Quit\n\n");
System.out.print("Enter selection: ");
input = keyboard.nextLine();
int choice = Integer.parseInt(input); // convert to int
String parameters, command, message;

switch (choice)
{
    case 1: // set RTC time
        System.out.print("Enter hhmmss: ");
        parameters = keyboard.nextLine();
        command = "!st" + parameters;
        message = "[0A" + command + "]";
        os.writeBytes(message);
        break;
    case 2: // set RTC date
        System.out.print("Enter wmmddyy (w: 1=Mon, 2=Tue, ..., 7=Sun: ");
        parameters = keyboard.nextLine();
        command = "!sd" + parameters;
        message = "[0A" + command + "]";
        os.writeBytes(message);
        break;
    case 3: // set display brightness
        System.out.print("Enter value (000 = automatic or value between 025 and 090): ");
        parameters = keyboard.nextLine();
        command = "!sb" + parameters;
        message = "[0A" + command + "]";
        os.writeBytes(message);
        break;
    case 4: // get HW serial number
        command = "!?n";
        message = "[0A" + command + "]";
        os.writeBytes(message);
        break;
    case 5: // get firmware id and version number
        command = "!?v";
        message = "[0A" + command + "]";
        os.writeBytes(message);
        break;
    case 6: // get HW status
        command = "!?h";
        message = "[0A" + command + "]";
        os.writeBytes(message);
        break;
    case 7: // get display configuration
        command = "!?z";
        message = "[0A" + command + "]";
        os.writeBytes(message);
        break;
    case 8: // get time
        command = "!?t";

```

```

        message = "[0A" + command + "];
        os.writeBytes(message);
        break;
case 9: // get date
    command = "!!d";
    message = "[0A" + command + "];
    os.writeBytes(message);
    break;
case 10: // get operational status
    command = "!!o";
    message = "[0A" + command + "];
    os.writeBytes(message);
    break;
case 11: // get error status
    command = "!!e";
    message = "[0A" + command + "];
    os.writeBytes(message);
    break;
case 12: // LED test
    command = "!!l";
    message = "[0A" + command + "];
    os.writeBytes(message);
    break;
case 13: // get number of failed LEDs
    command = "!!e";
    message = "[0A" + command + "];
    os.writeBytes(message);
    break;
case 14: // display sample text
    command = "!ps01!m1!cl1!dw"; // Send to address A; page store 01; mode 1; clear line
1; wait for effects
        command = command + "!dl1jpL"; // Display line 1; jump w/text appears instantly; left
justify
        command = command + "<\n1111111111111111>"; // String of ones
        command = command + "!dp01!cl2!dw"; // Display pause 001 second; clear line 2; wait
for effects
        command = command + "!dl2S1L"; // Display line 2; scroll one time to blank; left
justify;
        command = command + "<\n2222222222222222>"; // String of twos
        command = command + "!dp01!pl01"; // Display pause 001 second; set page list order to
page 1
        message = "[0A" + command + "];
        os.writeBytes(message);
        break;
case 15: // send POLL command
    command = "!!p";
    message = "[0A" + command + "];
    os.writeBytes(message);
    break;
case 16: // Delete all pages and clear the display
    command = "!!pr";
    message = "[0A" + command + "];
    os.writeBytes(message);
    break;
case 17: // reboot display
    command = "!!zb";

```

```

        message = "[0A" + command + "];
        os.writeBytes(message);
        break;
    case 18: // exit and quit
        System.out.println("Program Terminated.");
        break;
    default: // shouldn't happend
        System.out.println("Menu choice not found -- try again!");
        break;
}
if ((choice >= 1) && (choice <= 16))
{
    // keep on reading from/to the socket till we receive a response from the server,
    // once we received ']' (stop char) then we want to break.
    byte ch; int i = 0;
    byte [] responseLine = new byte [80];
    System.out.print("Server Response: ");
    while ((ch = is.readByte()) != -1) {
        responseLine[i++] = ch;
        System.out.printf("%c", responseLine[i-1]);
        if (responseLine[i-1] == ']') {
            break;
        }
    }
    System.out.println();
}
else if (choice == 17)
{
    System.out.println("Rebooting display ....");
    continue;
}
else if (choice == 18) break; // Exit and Quit
else continue; // if none of the menu choices, display menu and get again
}

// clean up: close input stream, output stream and socket
os.close();
is.close();
ledSocket.close();
} catch (UnknownHostException e) {
    System.err.println("Trying to connect to unknown host: " + e);
} catch (IOException e) {
    System.err.println("IOException: " + e);
}
}
}
}

```

Appendix D – Source Code for Dr. Brown’s ETA Calculations

AlIETA.java

```
package com.mymobed.clients.wave;

import java.sql.*;
import java.util.*;
import java.util.logging.Logger;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.sql.DataSource;

public class AlIETA {

    private int tol = 400;
    private int headTol = 30;

    /*
     * The ids of the breadcrumbs in the current table that are closest to Forden station.
     */
    private int[] targetBC = new int[]{2045,2209,5797,5934,7949,11449};
    private Route[] routes = new Route[6];
    private String[] routeNames = new String[]{"101 BROOKLYN","102 UNIVERSITY","103
CENTRAL","104 EAST","105 MEDICAL CENTER","106 WEST"};

    private String getAllLocations = "SELECT VI.Latitude as lat, VI.Longitude as lon, VI.Heading as
head, CONVERT(varchar, DATEADD(ss, VI.UpdateTime, '01/01/1970'), 21) AS report_time,
VI.RouteName AS line_id, VI.StopName AS NextStop FROM VehicleInfo VI WHERE VI.RouteID != '0'
AND VI.RouteID != '1000' AND DATEDIFF(second,'1/1/1970',GETDATE())-VI.UpdateTime <= 90 and
RouteName like '10%' order by RouteName";

    public String getAlIETA(){
        /*
         * The field where is an array of 6 Vectors, one for each of the routes 101 - 106.
         */
        Vector[] where = new Vector[6];
        Connection conn = null;
        StringBuffer sb = new StringBuffer();

        /*
         * First use the query getAllLocations to find all entries in the VehicleInfo table that are
current
         * and belong to one of the routes 101 - 106.
         * Since one route may have more than one bus, the array where holds a Vector of
Location objects
         * for each route.
         */

        try {
            //waveDS.setLoginTimeout(5);
            conn = waveDS.getConnection();
            //theLogger.info("Connection is null: "+(conn == null));
            Statement s = conn.createStatement();
            //theLogger.info("Statement is null: "+(s == null));
```

```

ResultSet r = s.executeQuery(getAllLocations);
//theLogger.info("ResultSet is null: "+(r == null));

while(r.next()){
    String routeName = r.getString("line_id");
    //theLogger.info("routeName = "+routeName);
    int routeIndex = indexOfRoute(routeName);
    //theLogger.info("routeIndex = "+routeIndex);
    if(where[routeIndex] == null) where[routeIndex] = new Vector();
    Location L = new
    Location(r.getInt("lat"),r.getInt("lon"),r.getInt("head"));
    where[routeIndex].add(L);
}
}
catch(Exception e){
    theLogger.severe("Error getting all ETA. Message: "+e.getMessage());
    //return "Error getting all ETA. Message: "+e.getMessage();
}
finally{
    try{conn.close();}
    catch(Exception ex){}
}

/*
 * For each route i (0<=i<6) we look at the Vector where[i] and calculate the ETA
 * for each Location object in the Vector. Report the smallest.
 * If where[i] == null, that means we did not get current location of any bus on that route.
 * In that case we estimate time based on schedule.
 */

for(int i=0; i<6; i++){
    int minTime = 1000000;
    Vector v = where[i];
    if(v == null){
        int time = getScheduledETA(i);
        sb.append(routeNames[i]);
        sb.append("*");
        sb.append(time);
        sb.append(',');
        continue;
    }
    for(int j = 0; j < v.size(); j++){
        Location L = (Location)v.get(j);
        int[] closest = getClosestBC(i+1,L);
        int time = routes[i].estimateTime(closest[0], closest[1]);
        if(time < minTime) minTime = time;
    }
    sb.append(routeNames[i]);

    if(minTime > 3600){
        sb.append("*");
        sb.append(getScheduledETA(i));
    }
    else {
        sb.append(',');
        sb.append(minTime);
    }
}

```



```

        }
        sb.append(',');
    }
    return sb.toString();
}

private int getScheduledETA(int indexOfRoute){
    Calendar c = Calendar.getInstance();
    int min = c.get(Calendar.MINUTE);
    if(min > 30) return (90 - min)*60;
    return (30 - min)*60;
}

private int[] getClosestBC(int routeId, Location where){
    /*
    * The Location object where has lat,lon, and heading info of a bus on route with routeId.
    * This method is going to find the breadcrumb that is closest to the current Location where.
    * For closest, we cannot use only lat and lon, but must also consider heading. This is because
    * bus might go by the same location going in different directions.
    * We do not expect the heading of where to exactly match a heading in the breadcrumb table,
    * so we look for headings within a certain tolerance, headTol, set at the top of this class.
    *
    * The three queries below correspond the the following cases:
    * query1: where.heading + headTol > 360
    * query2: where.heading - headTol < 0
    * query3: else
    *
    * When we have found the closest breadcrumb to the current location where, we return an
    * array int[2] with the id of the breadcrumb and timestamp (seconds from epoch) of breadcrumb.
    */
    String query1 = "select id, lat, lon, heading, datetime from waveBC where bc_id = ? and\n"
        + "abs(lat-?) < ? and abs(lon-(?)) < ? and ((? < heading) or (heading < (?%360)))";
    String query2 = "select id, lat, lon, heading, datetime from waveBC where bc_id = ? and\n"
        + "abs(lat-?) < ? and abs(lon-(?)) < ? and ((? < heading) or (heading < ?))";
    String query3 = "select id, lat, lon, heading, datetime from waveBC where bc_id = ? and\n"
        + "abs(lat-?) < ? and abs(lon-(?)) < ? and (? < heading) and (heading < ?)";

    int wlat = where.getLat();
    int wlon = where.getLon();
    int whead = where.getHeading();

    Connection conn = null;
    try{
        conn = mobedDS.getConnection();
        PreparedStatement ps = null;
        if((whead + headTol) > 360) ps = conn.prepareStatement(query1);
        else if((whead - headTol) < 0) ps = conn.prepareStatement(query2);
        else ps = conn.prepareStatement(query3);

        ps.setInt(1,routeId);
        ps.setInt(2, wlat);
        ps.setInt(3, tol);
        ps.setInt(4, wlon);
        ps.setInt(5, tol);
    }
}

```

```

        int param = whead - headTol;
        if((whead - headTol) < 0) param = 360 + whead - headTol;
        ps.setInt(6, param);
        ps.setInt(7, whead + headTol);
        ResultSet r = ps.executeQuery();

        int[] out = new int[2];
        int distance = 1000000;
        int count = 0;
        while(r.next()){
            count++;
            int id = r.getInt(1);
            int lat = r.getInt(2);
            int lon = r.getInt(3);
            int head = r.getInt(4);
            int time = r.getInt(5);

            int d = (wlat-lat)*(wlat-lat)+(wlon-lon)*(wlon-lon);
            if(d < distance){
                distance = d;
                out[0] = id;
                out[1] = time;
            }
        }
        return out;
    }
    catch(Exception e){
        theLogger.severe("Error finding closest. Message: "+e.getMessage());
        return new int[]{-1,-1};
    }
    finally{
        try{conn.close();}
        catch(Exception ex){}
    }
}

private int indexOfRoute(String name){
    /*
    * The VehicleInfo table identifies the route that a bus is on by name, 104 East, for example.
    * The getAllETA method keeps an array of Vectors associated with the six routes 101 - 106.
    * The helper method returns the array index associated with the route name.
    */
    if(name.startsWith("101")) return 0;
    if(name.startsWith("102")) return 1;
    if(name.startsWith("103")) return 2;
    if(name.startsWith("104")) return 3;
    if(name.startsWith("105")) return 4;
    if(name.startsWith("106")) return 5;
    return -1;
}

private DataSource waveDS;
private DataSource mobedDS;
private static Logger theLogger = Logger.getLogger(AllETA.class.getName());

```

```

public AllETA(){
    try{
        Context ctx = new InitialContext();
        waveDS = (DataSource)ctx.lookup("java:comp/env/jdbc/wave");
        mobedDS = (DataSource)ctx.lookup("java:comp/env/jdbc/mobed");
        for(int i=0; i<6; i++){
            Route r = new Route(i+1);
            /*
             * The Route and RouteInfo classes were designed to estimate the arrival time
             * at any stop on any route. The Route setTarget method sets the breadcrumbId
             * of the spot we are estimating arrival at.
             * This class is only estimating arrival at Forden Station, so we set the target
             * as the breadcrumb on each route that is closest to Forden Station.
             */
            r.setTarget(targetBC[i]);
            routes[i] = r;
        }
    } catch(Exception e){
        theLogger.severe("Failed to get DataSource");
        theLogger.severe(e.getMessage());
    }
}
}

```

Location.java

```
package com.mymobed.clients.wave;
```

```

/*
 * A Location object consists of a latitude (lat), a longitude (lon), and a heading.
 * Lat/Lon coordinates in the AVL database are stored as integers,
 * so lat and lon are int for compatibility. Heading is an int between 0 and 360,
 * degree clockwise from north.
 */
public class Location {

    public Location(int lat, int lon, int heading){
        this.lat = lat;
        this.lon = lon;
        this.heading = heading;
    }
    private int lat;
    private int lon;
    private int heading;

    public int getLat() {
        return lat;
    }
    public void setLat(int lat) {
        this.lat = lat;
    }
    public int getLon() {
        return lon;
    }
    public void setLon(int lon) {

```

```

        this.lon = lon;
    }
    public int getHeading() {
        return heading;
    }
    public void setHeading(int heading) {
        this.heading = heading;
    }
}

```

Route.java

```
package com.mymobed.clients.wave;
```

```

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.Hashtable;
import java.util.logging.Logger;

```

```

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.sql.DataSource;

```

```
public class Route {
```

```
/*
```

```
* The query getStopsAndBC was used by an app that calculated ETA to any stop.
```

```
* It is not used by AllETA
```

```
*/
```

```

        private String getStopsAndBC = "select stop.name,bc.breadcrumbid from TStop as
stop,TPatStopVarBCPTR as bc where stop.ID = bc.StopVarId and bc.PatID = ?";

```

```
/*
```

```
* The breadcrumb table has location, heading, and timestamp information at points along a route.
```

```
* The breadcrumb timestamps are a few seconds apart, so each route has many (over 1000) breadcrumbs.
```

```
* The breadcrumbs were created by driving the route and gathering data, not stopping to pickup and
```

```
* drop off passengers. So, the schedule allows an hour for the bus to make a circuit, but the total
```

```
* time to breadcrumb the route may be less than 30 minutes.
```

```
*
```

```
* Here is the basic algorithm.
```

```
* All times are in seconds.
```

```
* BC stands for breadcrumb.
```

```
* The scheduled time for each route is 1 hour = 3600 seconds.
```

```
* totalTime is the last BC timestamp - first BC timestamp, i.e. the total time to breadcrumb the route.
```

```
* est is the estimated time to Forden station using BC timestamps.
```

```
*
```

```
* As noted above, est is low, because the actual bus has to pickup and drop off.
```

```
* We will return est + error. Here is how error is calculated.
```

```
*
```

```
* if est <= 5 min = 300 seconds, error = 0. 300 is an arbitrary value, and the algorithm may
```

```
* be improved if this is adjusted. Variable ADJUSTMENT_THRESHOLD = 300. The idea is that if the
bus is close,
```

```
* then there are not many stops to slow it down.
```

```
*
```

```
* if est = totalTime, then the bus has just left, and it will be 3600 seconds before it returns.
```

```
* Remember that we return est + error, so error = 3600 - totalTime.
```

```

* est + error = totalTime + 3600 - totalTime = 3600.
*
* For est values between ADJUSTMENT_THRESHOLD=300 and totalTime, error is linear.
* We have two points on the graph of error: (ADJUSTMENT_THRESHOLD,0) and (totalTime, 3600 -
totalTime).
* Slope = (3600 - totalTime)/(totalTime - ADJUSTMENT_THRESHOLD).
*
* The constructor for Route uses the BC table to calculate totalTime and slope.
*/

    private int ADJUSTMENT_THRESHOLD = 300;

    private Hashtable stopsAndBC;
    public Hashtable getStopsAndBC(){return stopsAndBC;}

    private int routeId;
    public int getRouteId() {return routeId;}

    private String name;
    public String getName(){return name;}

    private int startBC;
    private int startTime;
    private int endBC;
    private int endTime;
    private int totalTime;
    private double slope;
    private int target = -1;
    private int targetTime;
    public void setTarget(int t){
        target = t;
        Connection conn = null;
        try {
            conn = datas.getConnection();
            PreparedStatement ps = conn.prepareStatement("select datetime from
threadcrumbs where id = ?");
            ps.setInt(1, target);
            ResultSet r = ps.executeQuery();
            r.next();
            targetTime = r.getInt(1);
        }
        catch(Exception e){
            theLogger.severe("Failed to set time for target = "+target);
            theLogger.severe("Message: "+e.getMessage());
        }
        finally {
            try{conn.close();}
            catch(Exception ex){}
        }
    }

    public int estimateTime(int bcId, int bcTime){
        if(bcId <= target) return adjustedEstimate(targetTime - bcTime);
        return adjustedEstimate(endTime - bcTime + targetTime - startTime);
    }

    public int adjustedEstimate(int est){

```

```

        if(est < ADJUSTMENT_THRESHOLD) return est;
        return est + Math.round((float)(slope*(est - ADJUSTMENT_THRESHOLD)));
    }
    private static Logger theLogger = Logger.getLogger(Route.class.getName());
    private DataSource datas;

    public Route(int routeId) throws Exception{
        //Get the name associated with this routeId
        name = (String)RouteInfo.routeIdsAndNames.get(new Integer(routeId));
        Connection conn = null;
        try {
            Context ctx = new InitialContext();
            datas = (DataSource)ctx.lookup("java:comp/env/jdbc/wave");
            conn = datas.getConnection();
            PreparedStatement ps = conn.prepareStatement(getStopsAndBC);
            ps.setInt(1, routeId);
            ResultSet r = ps.executeQuery();
            stopsAndBC = new Hashtable();
            while(r.next()){
                stopsAndBC.put(new Integer(r.getInt(2)), r.getString(1));
            }
            ps = conn.prepareStatement("select id,datetime from tbreadcrumbs where id =
(select min(id) from tbreadcrumbs where bc_id = ?)");
            ps.setInt(1, routeId);
            r = ps.executeQuery();
            r.next();
            startBC = r.getInt(1);
            startTime = r.getInt(2);
            ps = conn.prepareStatement("select id,datetime from tbreadcrumbs where id =
(select max(id) from tbreadcrumbs where bc_id = ?)");
            ps.setInt(1, routeId);
            r = ps.executeQuery();
            r.next();
            endBC = r.getInt(1);
            endTime = r.getInt(2);
            totalTime = endTime - startTime;
            slope = (3600.0 - totalTime)/(double)(totalTime -
ADJUSTMENT_THRESHOLD);//See comments at top of class
        }
        catch(Exception e){
            throw e;
        }
        finally {
            try{conn.close();}
            catch(Exception ex){}
        }
    }
}

```

RouteInfo.java

```
package com.mymobed.clients.wave;
```

```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
```

```

import java.util.*;
import java.util.logging.Logger;

import javax.naming.Context;
import javax.naming.InitialContext;
import javax.sql.DataSource;

public class RouteInfo {
    /*
     * This class holds information pulled from the Wave AVL database.
     * Cache here to cut down on DB queries
     *
     * There are two route IDs used in the AVL DB.
     * Three columns of the TPat table are ID, Name, RouteId
     * I am not sure where RouteId is used. Routes in the TBreadcrumb table (BC)
     * are identified by the number in the ID field of TPat.
     *
     * Neither of these route ids are used in the VehicleLocation table!
     * We have to identify route by RouteName in that table.
     *
     * NOTE: this class was used in a Web app that let the user select a route by name,
     * then select any stop on that route. It then calculated ETA to that stop.
     * The class Route uses the the Hashtable routeIdsAndNames to get the name from the id.
     * AllETA makes no other use of this class.
     */
    private static String getRouteNames = "select Name,ID from TPat";
    private static String getCurrentRoutes = "SELECT CONVERT(varchar, DATEADD(ss,
VI.UpdateTime, '01/01/1970'), 21) AS report_time, VI.RouteName AS line_id FROM VehicleInfo VI
WHERE VI.RouteID != '0' AND VI.RouteID != '1000' AND DATEDIFF(second,'1/1/1970',GETDATE())-
VI.UpdateTime <= 90";

    public static Hashtable routeNamesAndIds;
    public static Hashtable routeIdsAndNames;
    private static Logger theLogger = Logger.getLogger(RouteInfo.class.getName());
    private static DataSource datas;

    static{
        Connection conn = null;
        try {
            Context ctx = new InitialContext();
            datas = (DataSource)ctx.lookup("java:comp/env/jdbc/wave");
            conn = datas.getConnection();
            PreparedStatement ps = conn.prepareStatement(getRouteNames);
            ResultSet r = ps.executeQuery();
            routeNamesAndIds = new Hashtable();
            routeIdsAndNames = new Hashtable();
            while(r.next()){
                String name = r.getString(1);
                int id = r.getInt(2);
                routeNamesAndIds.put(name, new Integer(id));
                routeIdsAndNames.put(new Integer(id), name);
            }
        }
        catch(Exception e){
            theLogger.severe("Could not load route names");
            theLogger.severe("Message: "+e.getMessage());
        }
    }
}

```

```

    }
    finally {
        try{conn.close();}
        catch(Exception ex){ }
    }
}

public static Hashtable getCurrentRoutes(){
    Connection conn = null;
    Hashtable out = new Hashtable();
    try {
        conn = datas.getConnection();
        PreparedStatement ps = conn.prepareStatement(getCurrentRoutes);
        ResultSet r = ps.executeQuery();

        while(r.next()){
            String route = r.getString("line_id");
            //Ugly KLUDGE to get only routes 101 - 106
            if(!route.contains("10")) continue;
            Integer id = (Integer)routeNamesAndIds.get(route);
            if(!out.containsKey(id)) out.put(id, route);
        }
        return out;
    }
    catch(Exception e){
        theLogger.severe("Could not get current routes");
        theLogger.severe("Message: "+e.getMessage());
        return out;
    }
    finally {
        try{conn.close();}
        catch(Exception ex){ }
    }
}
}

```